

UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR



Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Aplicación web progresiva para una red social de aficionados a los videojuegos

Juan de Lis Aguirregomezcorta
Tutor: Miguel Ángel Mora Rincón

JULIO 2020

Aplicación web progresiva para una red social de aficionados a los videojuegos

AUTOR: Juan de Lis Aguirregomezorta

TUTOR: Miguel Ángel Mora Rincón

GHIA (Grupo de Herramientas Interactivas Avanzadas)

Dpto. de Ingeniería Informática

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Julio de 2020

Resumen

Actualmente, el uso de las redes sociales y los videojuegos está en continuo aumento, especialmente entre los jóvenes, que han cambiado por completo su forma de relacionarse y entretenerse debido a la utilización de las nuevas tecnologías. Es por ello, que este Trabajo Fin de Grado tiene como objetivo el desarrollo de una aplicación web que ofrezca a los aficionados a los videojuegos un lugar en la red donde relacionarse e intercambiar publicaciones de texto sobre videojuegos.

Para ello, en primer lugar, se ha realizado un análisis inicial, mediante encuestas a posibles futuros usuarios y un análisis competitivo de otras redes sociales actuales, con la intención de obtener los requisitos iniciales de la aplicación. En segundo lugar, se ha llevado a cabo una fase de diseño en la cual se detalla la arquitectura BaaS (*Backend as a Service*) del sistema y el uso del proveedor externo Firebase para ello, además del patrón de diseño MVVM (Modelo-Vista-VistaModelo) que utilizaremos en el *framework* Nuxt.js para el *frontend*. En tercer lugar, se ha realizado el desarrollo de la aplicación, comenzando con la creación del proyecto en Nuxt y su integración con el proveedor externo de *backend*, seguido de la implementación del modelo de datos con Vuex y de la interfaz de usuario con Vue y Vuetify. Por último, se describen las pruebas en local de los diferentes módulos y de su integración, así como la capacidad de visualizar estadísticas de uso a través de Firebase.

Con todo ello, el proyecto ha finalizado satisfactoriamente con la puesta en producción de la aplicación *GamersVoice*, la nueva red social para los aficionados al mundo de los videojuegos que les permite intercambiar publicaciones breves de texto.

Palabras clave

Redes sociales, Videojuegos, Aplicaciones Web, BaaS, Firebase, MVVM, Nuxt.js, Vue.js

Abstract

Currently, the use of social networks and video games is constantly increasing, especially among young people, who have completely changed their way of socializing and entertaining themselves, due to the use of new technologies. For that reason, this Final Degree Project aims to develop a web application that offers people interested in videogames a place online where they are able to interact and exchange text publications about videogames.

For the development of the web application, firstly, an analysis has been carried out, through surveys of potential future users and a competitive analysis of other current social networks, with the intention of obtaining the initial requirements of the application. Secondly, a design phase has been carried out, which details the BaaS (*Backend as a Service*) architecture of the system with the use of the external provider Firebase for it and also the MVVM (Model-View-ViewModel) design pattern that we will use in the Nuxt.js framework for the frontend. Thirdly, the application has been developed, starting with the creation of the project in Nuxt and its integration with the external backend provider. Then followed by the implementation of the data model with Vuex and the user interface with Vue and Vuetify. Finally, the local testing of the different modules and their integration are described, as well as the ability to view usage statistics through Firebase.

With all this, the project has successfully ended with the production deploy of the GamersVoice application, the new social network for those interested in the world of videogames, which allows them to exchange brief text publications.

Keywords

Social networks, Videogames, Web applications, BaaS, Firebase, MVVM, Nuxt.js, Vue.js

GLOSARIO

API	Application Programming Interface
Avatar	Representación gráfica de un usuario en el mundo virtual
BaaS	Backend as a Service
Backend	Capa de acceso a datos o servidor
DOM	Document Object Model
Framework	Conjunto de artefactos o módulos software que facilitan el desarrollo de software a través de librerías y código reutilizable
Frontend	Capa de presentación o interfaz de usuario
Getters	En Vuex, propiedades calculadas que se guardan en caché y se recalculan automáticamente cuando es necesario
MVVM	Modelo Vista Vista-Modelo
NPM	Node Package Manager
PWA	Progressive Web Apps
SDK	Software Development Kit
SPA	Single Page Application

INDICE

1. INTRODUCCIÓN.....	1
1.1 Motivación	1
1.2 Objetivos	1
1.3 Organización de la memoria	2
2. ESTADO DEL ARTE.....	3
2.1 Redes sociales y videojuegos.....	3
2.1.1 Aproximación a las redes sociales	3
2.1.2 Los videojuegos en las redes sociales.....	3
2.2 Desarrollo nativo vs multiplataforma	4
2.3 Aplicaciones web.....	5
2.3.1 Definición.....	5
2.3.2 Progressive Web Apps (PWA)	5
2.3.3 Single Page Application (SPA)	6
3. ANÁLISIS.....	7
3.1 Descripción y alcance de la aplicación	7
3.2 Metodología.....	7
3.3 Encuestas y análisis competitivo	8
3.4 Definición de la aplicación	9
3.4.2 Requisitos funcionales	10
3.4.3 Requisitos no funcionales	11
3.4.4 Diagrama de casos de uso	11
4. DISEÑO.....	13
4.1 Descripción del sistema	13
4.1.1 Arquitectura del sistema	13
4.1.2 Patrón de diseño MVVM	13
4.1.3 Diagrama de clases	15
4.2 Firebase	15
4.2.1 Introducción	15
4.2.2 Firebase Hosting	16
4.2.3 Firebase Authentication	16
4.2.1 Cloud Storage	16
4.2.2 Cloud Firestore	17
4.2.3 Cloud Functions	18
4.3 Node.js y Nuxt.js	18
4.3.1 Introducción	18
4.3.2 Vue.....	19
4.3.3 Vuetify	19
4.3.4 Vuex	19
5. DESARROLLO	21
5.1 Creación del proyecto en Nuxt	21
5.2 Modelo de datos (Vuex).....	23
5.2.1 Usuario autenticado (<i>user.js</i>).....	24
5.2.2 Usuario para mostrar (<i>userToShow.js</i>)	24
5.2.3 Resto de usuarios (<i>users.js</i>)	24
5.2.4 Publicaciones del usuario (<i>myposts.js</i>)	24

5.2.5	Publicaciones del resto de usuarios (<i>posts.js</i>)	24
5.3	Interfaz de usuario (Vue+Vuetify).....	25
6.	INTEGRACIÓN, PRUEBAS Y RESULTADOS	29
7.	CONCLUSIONES Y TRABAJO FUTURO	31
7.1	Conclusiones	31
7.2	Trabajo futuro	31
	REFERENCIAS.....	32
	ANEXOS.....	- 1 -
A.	Encuestas a futuros usuarios.....	- 1 -
B.	Principales casos de uso detallados	- 4 -
C.	Configuración y desarrollo de los servicios de Firebase	- 8 -
D.	Capturas de pantalla del desarrollo del proyecto	- 17 -
E.	Ejemplos de código de los módulos de Vuex.....	- 24 -
F.	Ejemplos de código de la interfaz (Vue y Vuetify)	- 35 -
G.	Capturas de pantalla de la versión móvil	- 39 -
H.	Generación y despliegue de la aplicación.....	- 45 -
I.	Herramienta de uso y facturación de Firebase	- 48 -

INDICE DE FIGURAS

FIGURA 2-1:	ARQUITECTURA BÁSICA DE UNA APLICACIÓN WEB.....	5
FIGURA 2-2:	DIFERENCIAS EN EL FUNCIONAMIENTO ENTRE SPA Y MPA	6
FIGURA 3-1:	CICLO DE VIDA INCREMENTAL.....	8
FIGURA 3-2:	DIAGRAMA DE CASOS DE USO.....	12
FIGURA 4-1:	DIAGRAMA DE LOS COMPONENTES DEL PATRÓN MVVM	14
FIGURA 4-2:	DIAGRAMA DE CLASES	15
FIGURA 4-3:	PRODUCTOS OFRECIDOS POR FIREBASE	16
FIGURA 4-4:	ESQUEMA DE LA ARQUITECTURA Y FLUJO DE VUEX	20
FIGURA 5-1:	ESQUEMA DE LAS VISTAS EN NUXT	22
FIGURA 5-2:	EJEMPLO DEL MÉTODO <i>onSnapshot()</i>	23
FIGURA 5-3:	VISTA INICIAL DE LA VERSIÓN WEB.....	25
FIGURA 5-4:	VISTA DEL ACCESO/REGISTRO CON FIREBASEUI.....	26
FIGURA 5-5:	VISTA DEL PERFIL DEL USUARIO AUTENTICADO	26
FIGURA 5-6:	VISTA DEL BUSCADOR DE USUARIOS.....	27
FIGURA 5-7:	VISTA DEL PERFIL DE OTRO USUARIO	27
FIGURA 5-8:	VISTA DE LAS PUBLICACIONES DE USUARIOS SEGUIDOS	28
FIGURA 6-1:	LANZAMIENTO DE LA APLICACIÓN EN LOCAL	29

INDICE DE TABLAS

TABLA 3-1:	ANÁLISIS COMPETITIVO	9
------------	----------------------------	---

1. INTRODUCCIÓN

1.1 Motivación

Es difícil pensar hoy en día en un mundo sin aplicaciones digitales ni comunicación a través de dispositivos electrónicos. Es por ello por lo que las redes sociales son un elemento esencial ya en la comunicación y relación entre las personas, especialmente entre los más jóvenes, que dedican una gran parte de su ocio en este tipo de aplicaciones. Tampoco es una novedad que los videojuegos son cada vez más populares en la sociedad actual, más aún ahora que hemos tenido que aprender a entretenernos sin salir de casa.

Así pues, parece bastante interesante la creación de una aplicación que combine estas dos cosas, ofreciendo a los aficionados a los videojuegos un lugar virtual donde relacionarse e intercambiar opiniones.

En el aspecto personal, este proyecto es interesante por la oportunidad que ofrece de estudiar y aprender el desarrollo de aplicaciones web y los distintos *frameworks* disponibles para hacerlo. Supone un reto personal y profesional, que seguro aportará mucho conocimiento y experiencia útil para el futuro fuera de la vida académica.

1.2 Objetivos

El objetivo principal del proyecto es desarrollar una red social que ofrezca a todo aquel que esté interesado en el mundo de los videojuegos un lugar en la red dónde conocer e interactuar con otras personas de gustos similares.

Es fundamental ofrecer un servicio diferente a los habituales hoy en día, los cuales buscan la sobreexcitación y atención constante del usuario, tratando de retenerle el máximo tiempo posible dentro de la aplicación. Este proyecto busca dar un enfoque diferente, a través de una aplicación sencilla pero potente, que cumpla siempre con los requisitos establecidos. Así pues, la aplicación pretende permitir a los usuarios relacionarse con otros y enviarse contenido en forma de texto.

Además, el desarrollo debe ser preciso y escalable, de forma que los recursos utilizados permitan un crecimiento tanto en los usuarios activos como en las funcionalidades desarrolladas en el futuro.

1.3 Organización de la memoria

La memoria se estructura en base a las diferentes fases que debe tener el desarrollo de un proyecto software. Por tanto, encontramos los siguientes apartados:

- Breve introducción con la motivación y objetivos del proyecto.
- Estado del arte, en el cual se estudian los conceptos y tecnologías relacionados con el proyecto.
- Análisis de la aplicación, dónde se describe la aplicación y la metodología utilizada y se extraen los requisitos a implementar.
- Diseño de la aplicación, con la arquitectura del sistema y las clases que lo compondrán, además de la introducción y explicación de las tecnologías que se utilizarán para implementar el *frontend* y el *backend*.
- Desarrollo de la aplicación y los diferentes componentes que se han implementado.
- Integración y pruebas de la aplicación durante y después de su implementación.
- Conclusiones obtenidas de la realización del proyecto y trabajo futuro que se podría realizar a continuación para mejorarlo.

2. ESTADO DEL ARTE

2.1 Redes sociales y videojuegos

2.1.1 Aproximación a las redes sociales

Las redes sociales han sido objeto de estudio en múltiples disciplinas, surgiendo entorno a ellas numerosas definiciones y teorías. Ureña afirma que una gran mayoría de autores coinciden en que una red social es “un sitio en la red cuya finalidad es permitir a los usuarios relacionarse, comunicarse, compartir contenido y crear comunidades”, o también puede ser definida como “herramienta de democratización de la información que transforma a las personas en receptores y en productores de contenidos” [1].

Una de las teorías más destacadas es la “teoría de los seis grados de separación”, propuesta inicialmente por el escritor húngaro Frigyes Karinthy y que fue descrita posteriormente por el sociólogo Duncan Watts [3]. Según esta teoría “cualquier persona en la Tierra puede conectarse con otra persona en el planeta a través de una cadena de conocidos que no tiene más de seis intermediarios” [2][3]. El origen de las redes sociales se relaciona con esta teoría ya que se basa en la idea de que cualquier persona estaría en posibilidad de conocer a otra persona del mundo a través de su red de contactos, de forma que uno de los fines que motivan su creación, aunque son varios, es principalmente la creación de un lugar de interacción virtual, en el que millones de personas alrededor del mundo se concentran con diversos intereses en común [1].

Las redes sociales se han encontrado en constante crecimiento y desarrollo desde su creación. A día de hoy, continúa aumentando su uso y el tiempo dedicado a ellas y han pasado a formar una parte muy importante de nuestras vidas según la Encuestas AIMC a Usuarios de Internet de 2020 [4].

2.1.2 Los videojuegos en las redes sociales

De la misma forma que las redes sociales, los videojuegos se encuentran en auge y se han convertido en un fenómeno lúdico relativamente reciente que forma parte importante de la cultura infantil y juvenil a nivel mundial [5]. Los estudios muestran que el 85% de los niños y adolescentes entre 9 y 16 años son usuarios de los videojuegos [6]. Sin embargo, su consumo no tiene éxito únicamente entre la población joven y es que los videojuegos se

están implantando con intensidad en el conjunto de la población española [7]. La mitad de los españoles mayores de 35 años (53,5 %) juega con videojuegos, pero además un estudio realizado por ISFE revela que la edad media de los jugadores en Europa se sitúa alrededor de los 35 años y es precisamente el grupo de entre los 25-34 años que ha tenido un crecimiento más rápido en los últimos años [8].

Independientemente de la edad de sus consumidores, no hay duda de que las redes sociales y los videojuegos han sufrido una evolución paralela y que los principales consumidores de los primeros son aquellos que también se encuentran presentes en la red. Es por ello por lo que en los últimos años encontramos que el mundo de los videojuegos se ha hecho un hueco importante también en las redes sociales. En este sentido, encontramos a los videojuegos muy presentes en redes sociales de temáticas generales, como son principalmente Twitch, Youtube y Twitter, donde los *gamers* se han hecho un gran hueco.

Por otro lado, durante los últimos años se han registrado la aparición de redes sociales especializadas en una actividad social o económica concreta, lo que permite satisfacer una necesidad inherente del ser humano de formar parte de grupos con características e intereses comunes [3]. Sin embargo, no encontramos numerosas ni populares redes sociales especializadas en el sector de los videojuegos, que den voz en internet a la comunidad *gamer*. Una comunidad acostumbrada a entretenerse a través de una pantalla pero que también necesita una forma similar de relacionarse en el entorno donde mejor se desenvuelven.

2.2 Desarrollo nativo vs multiplataforma

En el desarrollo de una aplicación software es necesario evaluar y decidir si la aplicación se implementará en un entorno nativo o multiplataforma, ya que se aprecian grandes diferencias en desarrollo y rendimiento [9].

En primer lugar, las aplicaciones nativas están específicamente codificadas para un sistema operativo concreto, de forma que deben utilizar el lenguaje y plataforma de desarrollo correspondiente, aumentando así el coste y tiempo necesario para la implementación en diferentes dispositivos. A cambio de ello, ofrecen una integración óptima con los recursos hardware y librerías del dispositivo, pudiendo ofrecer una mejor experiencia del usuario y un mayor rendimiento, aunque esto puede ser solamente significativo en aplicaciones muy exigentes.

Por otro lado, las aplicaciones multiplataforma se codifican una sola vez y se accede a ellas desde cualquier dispositivo a través de un navegador web, reduciendo los costes de desarrollo notablemente. Se facilitan también las tareas de mantenimiento y evolución, ya que los usuarios accederán siempre a la última versión subida en la red, sin necesidad de descargar o actualizar la aplicación con cada cambio o mejora.

2.3 Aplicaciones web

2.3.1 Definición

Un claro ejemplo de aplicaciones multiplataforma son las aplicaciones web, programas que interaccionan con el usuario a través de los navegadores web, ya sea a través de internet o de una red local. Ofrecen por tanto la capacidad de ejecutarse desde diferentes dispositivos, que además no necesitan ser muy potentes ni utilizar almacenamiento interno ya que la aplicación reside en un servidor externo. Además, como ya se ha visto, no provocan problemas de incompatibilidad de versiones entre usuarios pues siempre se tendrá acceso a la última versión publicada sin necesidad de instalaciones ni actualizaciones [10].

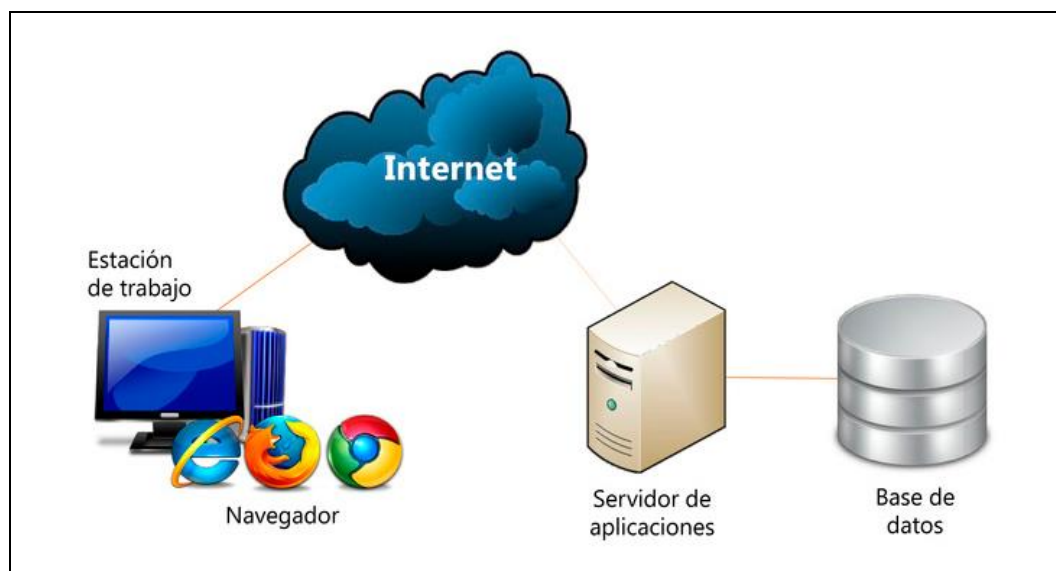


Figura 2-1: Arquitectura básica de una aplicación web

2.3.2 Progressive Web Apps (PWA)

Las aplicaciones PWA, presentadas por Google en 2015, consisten en aplicaciones web que tratan de ofrecer una experiencia de usuario similar a la ofrecida por aplicaciones nativas, pero a través de un entorno multiplataforma [11]. Permiten añadir la aplicación a

la pantalla de inicio de forma rápida y con el aspecto de una aplicación nativa, además de ofrecer la capacidad de funcionar *offline* almacenando contenido en la caché y de utilizar notificaciones *push*.

2.3.3 Single Page Application (SPA)

El complemento perfecto para las PWA son las SPA, aplicaciones que realizan una carga inicial de todo el código HTML, CSS y JavaScript en una sola página de forma que al navegar por la aplicación tan sólo se deba solicitar algo de contenido, sin necesidad de recargar una nueva página (como ocurre con las Multiple Page Application). Esto aporta fluidez y rapidez a la navegación, mejorando la experiencia de usuario y asemejándola a la de aplicaciones de escritorio [12].

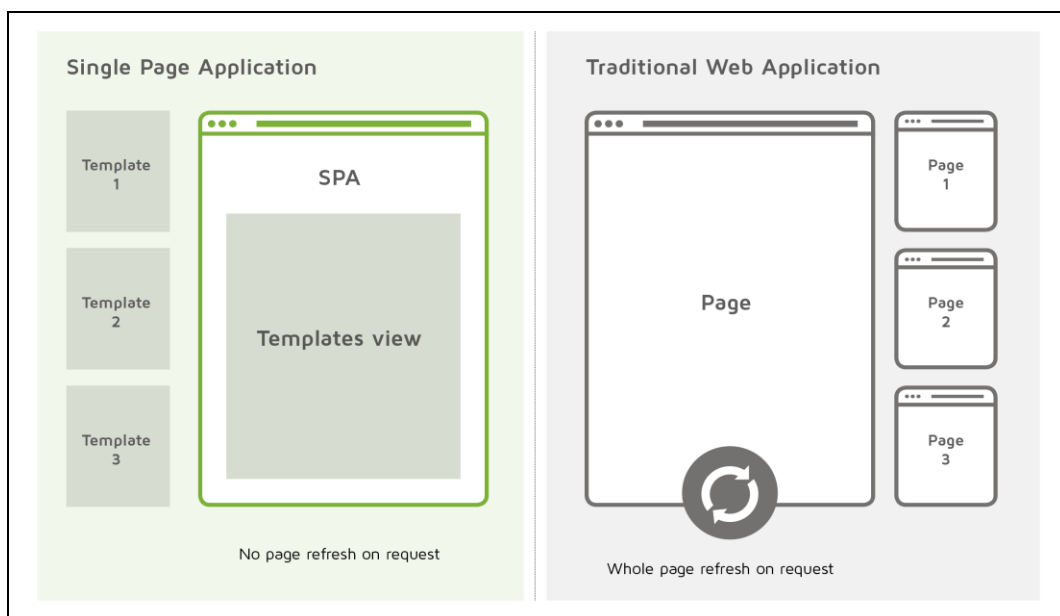


Figura 2-2: Diferencias en el funcionamiento entre SPA y MPA

A pesar de las grandes ventajas en velocidad, desarrollo y compatibilidad con móviles, esta implementación presenta algunos inconvenientes como pueden ser el SEO (Search Engine Optimization) o la necesidad de potenciar la seguridad frente a la ejecución de código malicioso en el cliente.

Los principales *frameworks* o librerías de JavaScript para implementar este tipo de aplicaciones son AngularJS, React o Vue.js, siendo este último el más reciente y el que combina las principales virtudes de los anteriores (virtual DOM, enlazamiento de datos, simplicidad, etc.), convirtiéndolo en el ideal para nuestro proyecto, como se irá explicando.

3. ANÁLISIS

3.1 Descripción y alcance de la aplicación

En este proyecto se va a implementar una aplicación de tipo red social que permita la relación y comunicación entre aficionados a los videojuegos, de una forma sencilla e intuitiva. Podrá ser utilizada por cualquier interesado que cuente con una cuenta de correo electrónico y que quiera interactuar con otros usuarios de gustos similares.

La aplicación, por tanto, estará compuesta por usuarios y sus publicaciones de texto, las cuales serán compartidas con el resto de los usuarios, que podrán valorarlas para así establecer una puntuación positiva y negativa para cada una de ellas.

Las relaciones entre usuarios serán unidireccionales, con un usuario origen que sigue a otro usuario destino. Dichas relaciones funcionan como una subscripción a los usuarios destino, de los cuales visualizarán en una página concreta las publicaciones que vayan subiendo.

En definitiva, el objetivo primordial de la aplicación será el de dar voz en la red a todos aquellos interesados en los videojuegos, para compartir en forma de texto sus experiencias, gustos, ideas o cualquier otro contenido relacionado con los videojuegos. Es por ello por lo que el nombre de la aplicación será *GamersVoice*.

3.2 Metodología

La metodología del proyecto debe tener en cuenta las características de este, como son la presencia de un único desarrollador, los plazos de entrega, futuros usuarios, etc. Así pues, se busca un desarrollo dinámico que sea adaptable a cambios y mejoras, pero con unos objetivos principales bien definidos para no perder el rumbo y poder cumplir los plazos.

Por todo ello, se ha decidido realizar una fase inicial de análisis, en la cual se llevarán a cabo un análisis competitivo y unas encuestas, para poder determinar unos requisitos iniciales base. Sobre dicha base se empezará a trabajar con un ciclo de vida incremental, que nos permite ir produciendo diferentes versiones operativas del producto a medida que nos acercamos al producto entregable final con toda la funcionalidad completa.

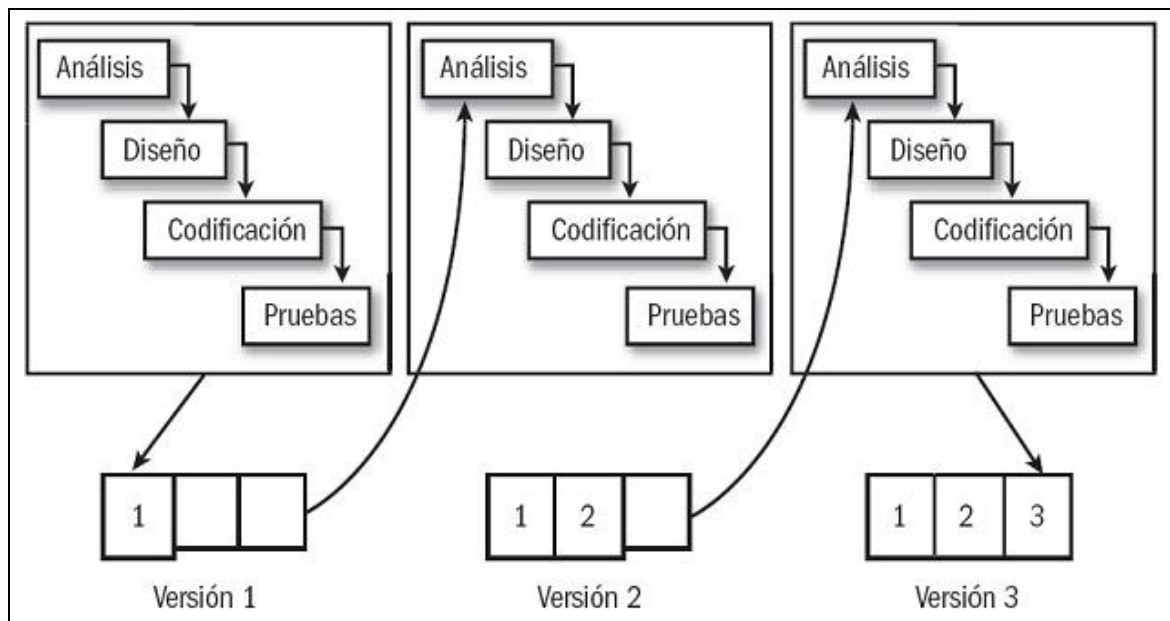


Figura 3-1: Ciclo de vida incremental

Las tres iteraciones sobre las que vamos a trabajar producirán una versión funcional con:

1. El registro/acceso del usuario y su perfil.
2. La consulta de otros usuarios y relaciones con ellos.
3. Las publicaciones del usuario y la interacción con las del resto de usuarios.

3.3 Encuestas y análisis competitivo

Como parte del análisis inicial se han realizado unas encuestas a posibles futuros usuarios y un análisis competitivo de aplicaciones similares ya operativas, ambas con la intención de aportar ideas para la extracción de los requisitos descritos en el próximo apartado.

Las encuestas se han utilizado para obtener información de los futuros usuarios y sus gustos o preferencias con respecto a la aplicación y posibles funcionalidades. Toda la información detallada de las preguntas y resultados se detallan en el anexo A.

El análisis competitivo consiste en una exploración por varias aplicaciones reales del sector con la intención de obtener ideas para nuestro proyecto. Por tanto, en cada una de ellas, se accederá e investigará para extraer tanto aspectos positivos como negativos y de ellos obtener futuras funcionalidades o posibles necesidades en el mercado que pueda satisfacer nuestra aplicación. Los resultados obtenidos son los siguientes:

Sistema URL:	Aspectos Positivos	Aspectos Negativos	Ideas/Mejoras para proyecto
TWITTER	<ul style="list-style-type: none"> • Lista de tendencias • Relación usuarios • App móvil 	<ul style="list-style-type: none"> • Publicaciones muy cortas 	<ul style="list-style-type: none"> • Relaciones entre usuarios de forma unidireccional • App móvil
LinkedIn	<ul style="list-style-type: none"> • Aplicación muy completa con varios menús y buscadores 	<ul style="list-style-type: none"> • Publicaciones no se diferencian bien entre sí 	<ul style="list-style-type: none"> • Diferenciar y separar bien las publicaciones
TUMBLR	<ul style="list-style-type: none"> • Publicaciones sencillas y cortas (Texto, link o multimedia) 	<ul style="list-style-type: none"> • Demasiado contenido al explorar 	<ul style="list-style-type: none"> • Publicaciones sencillas en forma y contenido • No sobrecargar de contenido para no saturar al usuario
CHOLLOMETRO	<ul style="list-style-type: none"> • Temperatura de los chollos para indicar éxito 	<ul style="list-style-type: none"> • Demasiados chollos en la pantalla (versión web) 	<ul style="list-style-type: none"> • Guardar valoración o <i>likes</i> de las publicaciones • Interfaz sencilla
TWITCH	<ul style="list-style-type: none"> • Menús lateral comprimible • Mucho contenido de videojuegos 	<ul style="list-style-type: none"> • Muy cargado de contenido • Enfocado a vídeos en directo 	<ul style="list-style-type: none"> • App más simple y enfocada a comunicación de texto entre usuarios

Tabla 3-1: Análisis competitivo

3.4 Definición de la aplicación

A partir de las ideas iniciales obtenidas en el análisis competitivo y las encuestas, de lo investigado sobre el sector con sus diferentes tecnologías y las diferentes fases de análisis de las iteraciones, se han determinado los requisitos y casos de uso de la aplicación.

Dichos requisitos estarán divididos en funcionales y no funcionales. Los primeros corresponden a acciones fundamentales del software, normalmente en forma de código y funciones concretas que ejecutan un proceso útil y directo para el usuario. Por otro lado, el segundo tipo se corresponde con características del sistema en general como pueden ser la seguridad, ejecución, estética, etc.

3.4.2 Requisitos funcionales

- REG-01: Los usuarios podrán crear una cuenta rellenando un sencillo formulario de registro con los campos: Correo, contraseña y nombre de usuario.
- LOG-01: Los usuarios se autenticarán con el correo y la contraseña introducidos en el registro o mediante su cuenta de Google.
- LOG-02: El sistema enviará un correo de restablecimiento de contraseña si el usuario lo solicita.
- CP-01: Los usuarios registrados dispondrán de una cuenta personal con sus credenciales de acceso, nombre de usuario y avatar.
- CP-02: Los usuarios podrán gestionar el contenido de su cuenta, visualizando y editando su imagen de avatar o cambiando su contraseña.
- CP-03: Los usuarios podrán eliminar su cuenta personal y todo el contenido relacionado con ella (publicaciones, valoraciones y relaciones).
- CP-04: Los usuarios no registrados no dispondrán de una cuenta personal y no podrán acceder a visualizar el contenido de los usuarios registrados.
- REL-01: Los usuarios serán capaces de establecer relaciones unidireccionales hacia otros usuarios (Origen=Seguidor y Destino=Seguido).
- REL-02: Los usuarios podrán eliminar las relaciones que previamente han creado hacia otros usuarios.
- PUB-01: Los usuarios podrán crear publicaciones nuevas, a partir de un mensaje de texto, junto con el usuario y la fecha de creación.
- PUB-02: Los usuarios dispondrán de una página con sus publicaciones y un buscador por contenido y fecha.
- PUB-03: Los usuarios podrán eliminar sus publicaciones.
- PUB-04: Los usuarios podrán valorar (Me gusta / No me gusta) publicaciones de otros usuarios.
- PUB-05: Las publicaciones de los usuarios seguidos se mostrarán cronológicamente en una página concreta, con un buscador por valoración y fecha.
- USR-01: Se podrá acceder a una página concreta con un buscador de usuarios, cuyos filtros serán el nombre de usuario y la relación hacia ellos.

- USR-02: Se podrá acceder al perfil de un usuario concreto, visualizando sus publicaciones con el mismo buscador por valoración y fecha de la página de publicaciones seguidas.
- USR-03: Tanto en la página con el buscador como en la página del perfil del usuario buscado, se podrá modificar la relación hacia dicho usuario.

3.4.3 Requisitos no funcionales

- SEG-01: Permitir el uso de la aplicación solo a usuarios autenticados.
- SEG-02: La consulta y modificación de la base de datos estará protegida por unas reglas de seguridad para evitar que un usuario modifique documentos de otros usuarios o que accedan usuarios no autenticados.
- SEG-03: Las contraseñas por pantalla se visualizarán ocultas con asteriscos y se almacenarán encriptadas.
- APR-01: El sistema debe tener un tiempo de aprendizaje para el usuario inferior a 15 minutos.
- INT-01: Interfaz sencilla, con colores oscuros para proteger la vista y que encajen con el estilo de los videojuegos.
- MULT-01: La aplicación será multiplataforma y permitirá su utilización a través de cualquier navegador estándar.
- SIM-01: Permitir acceso simultáneo a diferentes usuarios, sincronizando en tiempo real los cambios que realicen para no producir errores en dicho uso simultáneo.

3.4.4 Diagrama de casos de uso

A continuación, se muestra el diagrama de casos de uso definido para la aplicación, con un único actor principal que es el Usuario y sus diferentes acciones posibles, las cuales requieren de un registro y acceso inicial.

Por un lado, podemos ver los casos de uso relacionados con las publicaciones (creación, borrado, búsqueda o valoración). Por otro lado, tenemos los relacionados con otros usuarios (búsqueda, relación y visualización). Y, por último, los del propio perfil del usuario (cambiar contraseña o avatar).

4. DISEÑO

4.1 Descripción del sistema

4.1.1 Arquitectura del sistema

Siguiendo la línea de lo estudiado y analizado anteriormente, en nuestra aplicación web vamos a diferenciar claramente entre la parte del *frontend* (cliente) y la del *backend* (servidor), tratando de buscar una mayor modularidad en nuestro sistema que nos permita ir implementando los diferentes bloques con cierta independencia además de escoger tecnologías concretas que se adapten mejor a cada uno de ellos.

Por ello, y debido al alto coste de desarrollo que requiere diseñar e implementar un servidor propio, se ha optado por seguir una arquitectura BaaS (*Backend as a Service*), la cual consiste en prescindir de implementar íntegramente el *backend* de la aplicación y utilizar un proveedor externo, a través de la utilización de su API, que hace de puente entre el *frontend* de la aplicación y los servicios en la nube ofrecidos [13]. El proveedor escogido será Google Firebase, que se detallará más adelante junto con sus servicios ofrecidos, como puede ser el alojamiento de la aplicación, el sistema de autenticación de usuarios, las bases de datos NoSQL, las funciones en la nube o las estadísticas de uso y rendimiento.

Para la implementación del *frontend* utilizaremos el *framework* Nuxt.js (basado en Vue.js y Node.js), ya que nos permite fácilmente implementar una aplicación SPA (*Single Page Application*), que hace una carga inicial de todo el código estático de la aplicación y luego utiliza los servicios necesarios del proveedor BaaS, evitando la implementación de un servidor propio. En próximos apartados se detallarán todas las tecnologías utilizadas para dicha implementación.

4.1.2 Patrón de diseño MVVM

El patrón de diseño que se usa en Vue.js y que seguiremos en nuestro desarrollo es el MVVM (Modelo Vista Vista-Modelo), presentado al público en 2005 por John Gossman como una variación del MVC (Modelo Vista Controlador) [14], aunque realmente guarda también muchas similitudes con el patrón PM (Presentation Model) de Martin Fowler [15].

Al igual que el mencionado MVC, el patrón MVVM trata de conseguir una clara diferenciación entre la interfaz de usuario y su lógica a través de la estructuración en tres componentes fundamentales:

- Vista: Es la interfaz del usuario, centrada en mostrarle la información al usuario e interactuar con él.
- Modelo: Los datos y la lógica de negocio de la aplicación, así como la sincronización con el servidor externo para tener la información coherente y actualizada con el *backend*.
- Vista-Modelo: Es una abstracción de la vista que contiene toda su lógica y hace de intermediario entre ella y el modelo, interactuando con este último invocando a sus métodos y poniendo sus datos a disposición de la vista a través del enlazamiento de datos.

El componente vista-modelo separa la vista del modelo, permitiendo que este evolucione con independencia de la vista. Por tanto, las relaciones entre los componentes fluyen en una dirección: la vista tendrá referencias a la vista-modelo y esta actuará sobre el modelo, pero no al contrario, tal y como vemos en la siguiente figura.

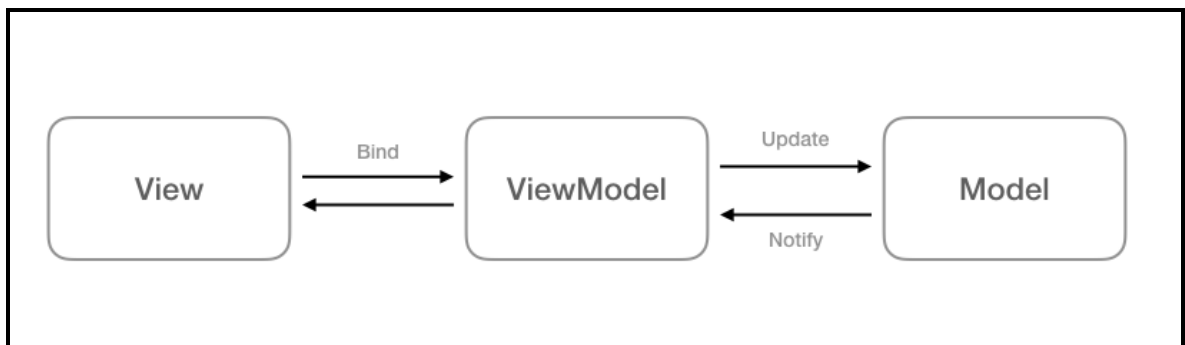


Figura 4-1: Diagrama de los componentes del patrón MVVM

Las principales ventajas que nos proporciona el desacople entre los mencionados componentes son la capacidad de desarrollar o probar los componentes de forma aislada y la posibilidad de hacerlo incluso utilizando diferentes tecnologías.

4.1.3 Diagrama de clases

A continuación, se muestra el diagrama de clases diseñado para satisfacer las funcionalidades planteadas:

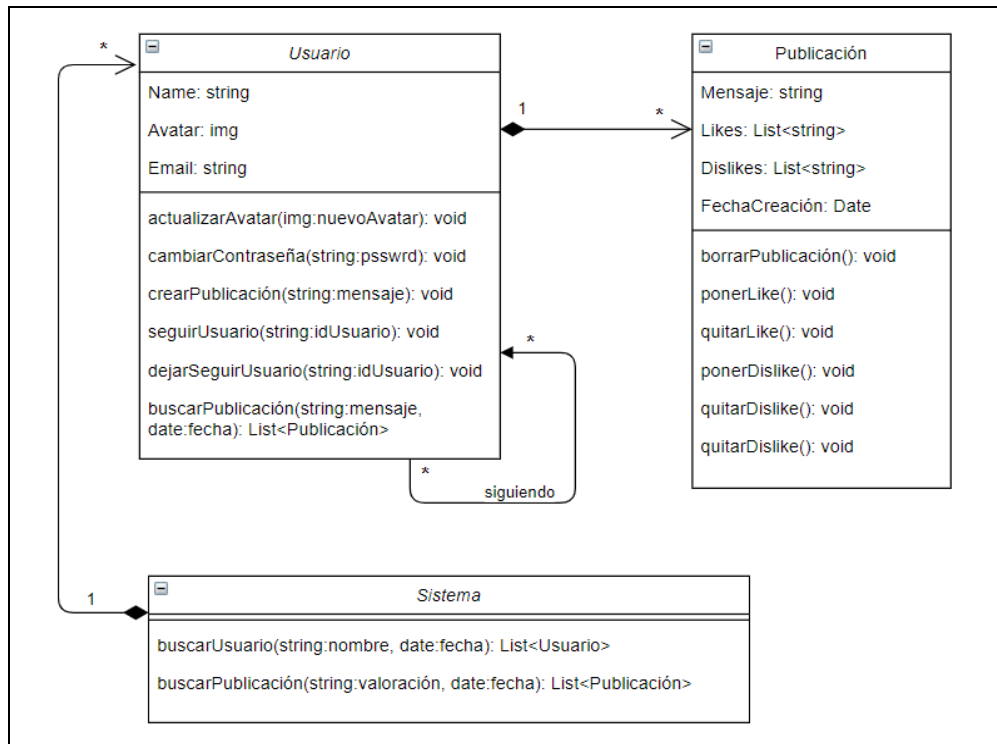


Figura 4-2: Diagrama de clases

4.2 Firebase

4.2.1 Introducción

Firebase, adquirida por Google en 2014, es una plataforma para el desarrollo de aplicaciones web y móviles, que proporciona las herramientas necesarias para poder desarrollar aplicaciones de alta calidad centrándonos en el lado del cliente, sin tener que preocuparnos por la compleja implementación de un servidor o base de datos propios [16].

Una de las principales ventajas de la utilización de este tipo de servicios es la mencionada posibilidad de prescindir de un desarrollo propio del *backend*, lo cual supondría un alto coste de esfuerzo y tiempo que no entra en el ámbito de este proyecto. Además, es fácilmente configurable e integrable en nuestro desarrollo del cliente con Nuxt.js a través de las SDK de los diferentes servicios utilizados, siguiendo las detalladas explicaciones de la documentación [17].

A continuación, se van a ir detallando los diferentes servicios de Firebase que se utilizarán en la aplicación, que serán Hosting, Authentication, Firestore, Storage y Functions, y sus respectivas configuraciones e implementaciones. Todo esto se complementa con las capturas de pantalla que se incluyen en el anexo C.



Figura 4-3: Productos ofrecidos por Firebase

4.2.2 Firebase Hosting

Proporciona el dominio y toda la infraestructura necesaria para alojar la aplicación web, garantizando siempre la transmisión segura de contenido mediante encriptación SSL (*Secure Sockets Layer*). La configuración es tan sencilla como indicar, en el fichero `firebase.json`, el directorio que se quiere implementar en el atributo “public” (en nuestro caso `dist`) y los archivos que se quieren evitar en el atributo “ignore”.

4.2.3 Firebase Authentication

Nos ofrece los servicios de *backend* e interfaz necesarios para llevar a cabo la autenticación de los usuarios, admitiendo el uso de contraseñas, números de teléfono o cuentas de Google, Facebook, Twitter, etc.

La interfaz ofrecida, `FirebaseUI`, es de código abierto y puede personalizarse para adaptarlo a nuestras necesidades o estilos de la aplicación. Para la aplicación, se ha importado la versión en español y se ha configurado para que permita el acceso únicamente a través de correo con contraseña o de cuenta de Google.

4.2.1 Cloud Storage

Es un servicio que permite un almacenamiento escalable en la nube de archivos como imágenes, audio, video o contenido de otro tipo generado por el usuario. Contiene reglas

de seguridad que pueden regular los accesos y modificaciones de dichos archivos en base a su nombre, tamaño, tipo de archivo u otros metadatos.

El servicio se ha utilizado para almacenar las imágenes de avatar de los usuarios, poniendo al archivo el id del usuario como nombre, para facilitar la búsqueda y gestión de los ficheros. Esto facilita la definición de las reglas de seguridad, que deben comprobar que el nombre del archivo es el id del usuario autenticado y que el fichero es de tipo imagen.

4.2.2 Cloud Firestore

Es la base de datos NoSQL en la nube, que permite almacenar datos de forma sencilla y escalable, además de mantenerlos sincronizados entre usuarios por medio de agentes de escucha en tiempo real. Utiliza colecciones, que contienen los documentos en los cuales se almacenan los datos a través de campos de varios tipos (strings, números, booleanos, fecha, referencia, arrays o mapas) y subcolecciones con otros documentos. Las consultas son muy eficientes y permiten recuperar datos en el nivel del documento, sin la necesidad de obtener la colección completa ni las colecciones anidadas.

Debido a que la eficacia aumenta con el uso de documentos cortos (incluso se limita el tamaño máximo de los documentos a 1 MB), se ha tratado de estructurar los datos en documentos sencillos pero que a la vez cumplan las funcionalidades de la aplicación, dando lugar a las siguientes tres colecciones:

- Usuarios: Con documentos cuyo nombre es el id (identificador) del usuario en Firebase Authentication y que almacenan en cadenas de caracteres su correo, nombre y avatar (referencia a la imagen almacenada en el Storage).
- Relaciones: Con documentos que almacenan la fecha de creación y los ids del usuario origen y del usuario destino.
- Publicaciones: Con documentos que almacenan el texto de la publicación, el usuario creador (id y nombre), fecha de creación y arrays con los ids de los usuarios que la han valorado (uno para votos positivos y otro para negativos).

Además, se han definido las reglas de seguridad que controlan el acceso y la modificación de los documentos de dichas colecciones. El primer requisito indispensable es el de estar autenticado en la aplicación. Los demás requisitos dependen de la colección:

- Usuarios: La creación y eliminación de un documento de usuario comprueba que el nombre del documento sea el id del usuario autenticado, evitando así poder crear o eliminar documentos de otros usuarios. Para la creación además se comprueba que no exista previamente el documento.
- Relaciones: La creación y eliminación de documentos de relaciones se permite si el usuario autenticado es el origen y no el destino de esta.
- Publicaciones: Para la creación y eliminación se comprueba que el campo del creador tenga el id del usuario. Se permite la modificación de documentos creados por otros usuarios siempre y cuando se modifiquen únicamente los arrays de valoraciones para añadir o quitar el id del usuario autenticado.

4.2.3 Cloud Functions

Permite almacenar código JavaScript en la nube y ejecutarse como respuesta a eventos generados por activadores de Firebase Authentication o Cloud Firestore, por ejemplo. Se proporciona además toda la infraestructura y mantenimiento necesarios, así como la escalabilidad y aumento de recursos según se requiera. La ventaja de guardar y ejecutar este código en el servidor es que aporta privacidad y seguridad al sistema, ya que aísla esta lógica del cliente evitando así alteraciones indeseadas.

Este servicio se ha utilizado en la aplicación para llevar a cabo todo el proceso de borrado de un usuario autenticado, de forma que al borrar el documento en Firestore de un usuario, se borren todos los documentos de publicaciones y relaciones que impliquen a dicho usuario, así como las valoraciones que haya podido hacer. También se debe eliminar su imagen del Cloud Storage y el usuario de Firebase Authentication.

4.3 Node.js y Nuxt.js

4.3.1 Introducción

Node.js es un entorno de programación orientado a eventos asíncronos para la ejecución de código JavaScript, complementado además con el sistema de gestión de paquetes npm que permite gestionar módulos de código ya desarrollado y dependencias de manera sencilla a través de comandos breves [18]. Se utilizará para ejecutar los comandos de creación del proyecto en Nuxt.js, de su compilación y subida al servidor (Firebase),

además del de ejecución de la aplicación en local, el cual permite probar y perfeccionar el producto en modo desarrollo antes de subirlo al servidor ya en modo producción.

Por su parte, Nuxt.js es un *framework* de JavaScript basado en las librerías oficiales de Vue.js (vue, vue.router y vuex) para el desarrollo de aplicaciones web modernas y potentes que se centra en facilitar el desarrollo, proporcionando una estructura clara al proyecto sin dejar de ofrecer una alta versatilidad y adaptabilidad en la implementación [19]. Nos ofrece la posibilidad de desarrollar una aplicación SPA prescindiendo de implementar un servidor propio (aunque también lo permite) y de añadir módulos o complementos muy fácilmente.

4.3.2 Vue

Vue es uno de los *frameworks* de JavaScript más populares, junto con el resto de los mencionados al comienzo. Como ya se ha explicado, está enfocado en el desarrollo de interfaces de usuario y SPAs, potenciando las capacidades de las vistas a través de la modificación de los valores mostrados o atributos de etiquetas HTML por enlace de datos, de la inserción de condicionales o bucles a la hora de mostrar contenido y de la invocación a métodos JavaScript o a acciones del modelo [20]. Por tanto, se comportará como VistaModelo en la arquitectura descrita para la aplicación, haciendo de intermediario entre la vista (Vuetify) y el modelo (Vuex).

4.3.3 Vuetify

Vuetify es una librería de componentes de interfaz para Vue desarrollada en 2016 y que sigue la especificación de diseño publicada por Google, “Material Design” [21]. Dispone de una gran librería de componentes visuales personalizables, desde elementos de formulario sencillos como botones o entrada de texto, hasta componentes más avanzados como menús desplegables o tarjetas de contenido. La organización de los elementos sigue el habitual sistema de cuadrículas, además de ofrecer la personalización de los elementos y sus tamaños en base a las dimensiones del dispositivo. Se utilizará para complementar el código HTML y Vue, actuando como Vista del modelo seguido.

4.3.4 Vuex

Vuex es una librería para aplicaciones Vue encargada de la gestión de los estados de la aplicación, es decir, de los datos de nuestro sistema y las funciones que los modifican [22]. Es el resultado de implementar el patrón de diseño Flux, usado por Facebook [23].

Esta arquitectura surge ante la necesidad de tener varios componentes visualizando o modificando el mismo estado y que no se comprometa la integridad o coherencia de los datos. Para ello, se extrae el estado de los componentes a un objeto único compartido, para que todos puedan acceder a su estado e invocar los métodos del objeto que lo modifican.

Por tanto, los componentes serán capaces de invocar las acciones (*actions*) definidas en el estado (*store*), en las cuales reside toda la lógica y llamadas al servidor externo (Firebase). Estas acciones ejecutarán los métodos que realizan los cambios (*mutations*) de las variables del estado (*state*). Dichas modificaciones provocan el renderizado de las vistas con el nuevo estado, ya que los componentes se encuentran vinculados con el estado por medio de observadores. Así pues, como vemos en la imagen siguiente, la arquitectura de Vuex tiene un flujo unidireccional cíclico:

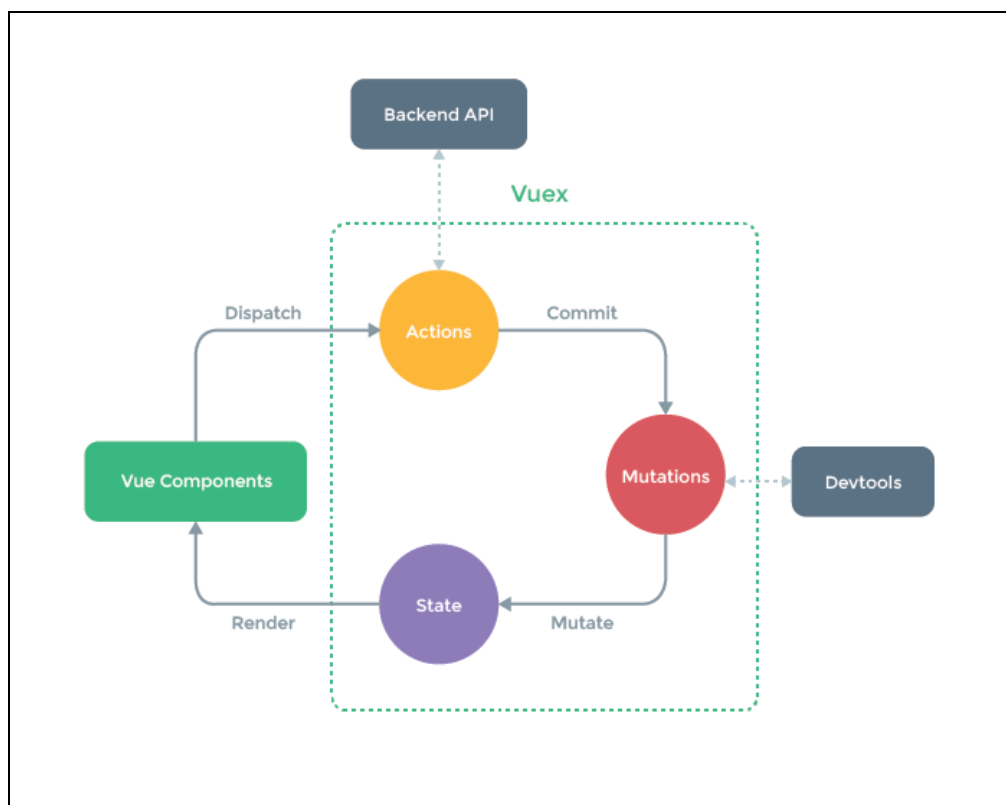


Figura 4-4: Esquema de la arquitectura y flujo de Vuex

Asimismo, los componentes no sólo pueden acceder a los valores del *state* para mostrarlos por pantalla o hacer cálculos con ellos, sino que también pueden acceder a propiedades calculadas a partir de ellos (*getters*), cuyos resultados se almacenan en caché y se actualizan únicamente cuando cambian las variables del estado de las que dependen.

5. DESARROLLO

5.1 Creación del proyecto en Nuxt

Planteado todo lo anterior, se va a explicar el desarrollo e implementación de la aplicación, junto con las diferentes tecnologías empleadas para ello. Asimismo, se complementará con capturas de pantalla incluidas en el anexo D.

La implementación de la aplicación pasa por comenzar creando el proyecto de Nuxt ejecutando el comando “`npx create-nuxt-app <project-name>`” y seleccionando algunos aspectos básicos de la aplicación: introducir nombre del proyecto, escoger lenguaje JavaScript, escoger gestor de paquetes npm, escoger *framework* de interfaz Vuetify.js, añadir módulo PWA, escoger herramienta de comprobación de código estático ESLint, escoger tipo de renderizado SPA y escoger *hosting* estático. Una vez configurado todo se genera el proyecto, instalando todos los elementos y dependencias indicados, y tendremos ya el esqueleto base sobre el que desarrollar la aplicación.

Para integrar la aplicación con el *backend* se añade Firebase a través del gestor npm para luego, en el fichero “`\services\fireinit.js`”, inicializarlo con la configuración del proyecto de Firebase y exportar los servicios de autenticación y bases de datos explicados anteriormente, que serán luego utilizados en la aplicación.

A continuación, se listan y explican los diferentes ficheros y directorios que componen la estructura básica, junto con las implementaciones realizadas en cada uno:

- `nuxt.config.js`: Fichero con la configuración personalizable de Nuxt, con algunos campos importantes como el modo (SPA), los módulos añadidos (PWA) o los recursos css que queremos declarar globalmente (Vuetify styling).
- `package.json`: Fichero que contiene la información básica de la aplicación (nombre, versión, descripción y autor), las dependencias de la misma y la lista de comandos esenciales como pueden ser “`nuxt`” para lanzar la aplicación en modo desarrollo en `localhost:3000` o “`nuxt build`” para compilar y construir la aplicación para producción.

- **Assets:** Contiene activos no compilados como, por ejemplo, un fichero JavaScript con métodos genéricos que interactúan con la base de datos para crear/eliminar el documento de una publicación y para eliminar el de un usuario.
- **Layouts:** Contiene el diseño que envolverá por defecto a las diferentes páginas de la aplicación. También se encuentra aquí la vista que se muestra en caso de producirse algún error.
- **Middleware:** Podemos definir aquí ficheros con código que se debe ejecutar antes de cargar una página. Para la aplicación se ha definido una función que comprueba que el usuario esté autenticado, y si no, le redirige al acceso/registro.
- **Pages:** Contiene las diferentes páginas de la interfaz de usuario por las que navegaremos y con las que se forma el enrutador. Se detallará su implementación en el apartado de la interfaz de usuario.

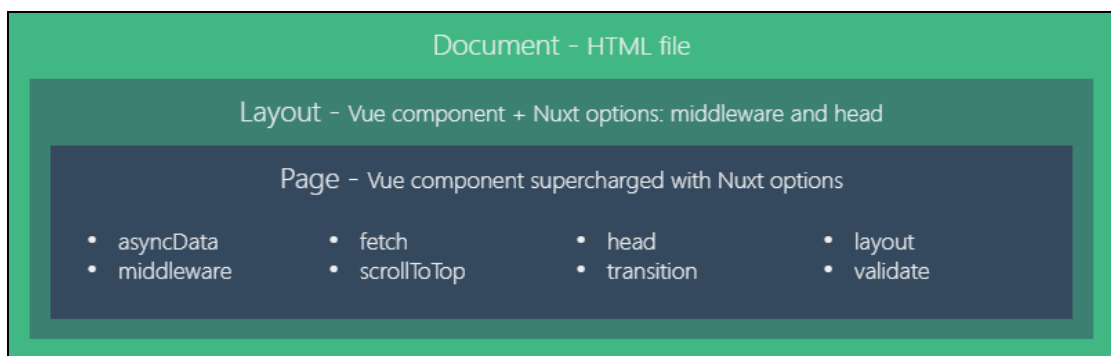


Figura 5-1: Esquema de las vistas en Nuxt

- **Plugins:** Contiene todos los complementos JavaScript que se ejecutarán antes de instanciar la aplicación. Aquí se ha introducido el fichero en el cual se registra la librería de Vuetify en Vue.
- **Static:** Contiene los ficheros usados por la aplicación y que permanecerán inmutables, entre los cuales están el icono de la aplicación o la imagen que se muestra por defecto como avatar en los usuarios que no han subido uno propio.
- **Store:** Contiene los ficheros JavaScript de Vuex, encargados del modelo de datos de la aplicación. Dichos ficheros corresponden a los diferentes módulos, cada uno con sus correspondientes *state*, *getters*, *mutations* y *actions*. Se describirán en profundidad en el siguiente apartado.

5.2 Modelo de datos (Vuex)

Los datos de la aplicación serán gestionados por el *store*, que los mantendrá siempre sincronizados con las bases de datos, escuchando los cambios producidos en esta y modificándola cuando sea necesario.

Es por eso por lo que todos los módulos del *store* comienzan importando métodos o servicios que nos ofrece Firebase, pues serán imprescindibles para la gestión de la información. Asimismo, las variables almacenadas en el *state* deberán ser similares o coherentes con las bases de datos, y se mantendrán sincronizadas con ellas a través del método *onSnapshot()* que nos ofrece Firebase para sus documentos, colecciones o consultas. Todos los módulos utilizarán dicho método para crear un agente de escucha y que, con cada creación, modificación o borrado del documento o colección correspondiente, se obtenga una instantánea y se ejecute la función indicada al método. Esto permite poner en escucha el *store* de forma que se actualice en tiempo real con los cambios producidos en la base de datos. El método permite incluso ver los cambios producidos entre instantáneas (*snapshot.docChanges()*).

```
db.collection("cities").where("state", "==", "CA")
  .onSnapshot(function(snapshot) {
    snapshot.docChanges().forEach(function(change) {
      if (change.type === "added") {
        console.log("New city: ", change.doc.data());
      }
      if (change.type === "modified") {
        console.log("Modified city: ", change.doc.data());
      }
      if (change.type === "removed") {
        console.log("Removed city: ", change.doc.data());
      }
    });
  });
```

Figura 5-2: Ejemplo del método *onSnapshot()*

El siguiente paso será profundizar en la composición de cada uno de los diferentes módulos de la aplicación: Usuario autenticado, resto de usuarios, usuario para mostrar, publicaciones del usuario autenticado y publicaciones del resto de usuarios. Las capturas de pantalla del código implementado se pueden ver en el anexo E.

5.2.1 Usuario autenticado (*user.js*)

Así pues, el módulo correspondiente al usuario autenticado se compone de un *state* que almacena la información del usuario (id, nombre, correo y avatar) junto con sus relaciones con otros usuarios (id de usuarios seguidores y seguidos). El único *getter* del módulo nos indica si el id de usuario almacenado es distinto de *null*, lo cual significa que está autenticado. Las *actions* del módulo nos permitirán cambiar o borrar la imagen del usuario en el Storage y su referencia en Firestore, empezar o dejar de seguir a otros usuarios, cerrar sesión y activar un agente de escucha para el servicio de autenticación de Firebase. Esta última es importante, ya que se activa para detectar cuando inicia y cierra sesión, activando y desactivando, respectivamente, los agentes de escucha correspondientes al usuario.

5.2.2 Usuario para mostrar (*userToShow.js*)

Este módulo corresponde a un usuario, diferente al autenticado, del cual se almacena tan sólo su información básica (id, nombre y avatar). Tan sólo contiene las acciones y mutaciones necesarias para mantener su información sincronizada con las bases de datos.

5.2.3 Resto de usuarios (*users.js*)

Para finalizar lo relacionado con los usuarios del sistema se ha implementado el módulo que contiene la información básica (id, nombre y avatar) de los que se mostrarán en el buscador de usuarios de la interfaz. Es por ello por lo que las acciones del módulo consisten en un buscador de usuarios a partir de los filtros introducidos y en un agente de escucha que también tendrá en cuenta dichos filtros de búsqueda. Se ha implementado un *getter* que da el número de usuarios almacenados y que se mostrarán por pantalla.

5.2.4 Publicaciones del usuario (*myposts.js*)

Respecto a las publicaciones, se ha implementado un módulo para las del usuario autenticado, de forma que se guarda el id, texto, fecha y valoraciones positivas y negativas de todas sus publicaciones. Dichas publicaciones tendrán también un buscador en la interfaz, lo que implica que las *actions* tanto de sincronización con la base de datos como de búsqueda deben filtrar las publicaciones con los parámetros introducidos.

5.2.5 Publicaciones del resto de usuarios (*posts.js*)

Para acabar con a las publicaciones, se ha implementado un módulo para las publicaciones que no son del usuario autenticado, de forma que se guarda el id, creador,

texto, fecha y valoraciones positivas y negativas de las publicaciones de un usuario concreto o de todos los usuarios seguidos por el autenticado. En este caso, la sincronización y búsqueda de publicaciones tendrá también en cuenta los filtros introducidos en el buscador, pero además debe tener en cuenta si se trata de las publicaciones de un usuario concreto o de todos los usuarios seguidos. Se han implementado también las acciones para poner o quitar la valoración de una publicación.

5.3 Interfaz de usuario (Vue+Vuetify)

Como ya se ha indicado, la implementación de la interfaz de usuario se ha realizado con Vue y Vuetify en los ficheros “.vue”, que se encuentran en los directorios *layout* y *pages*, los cuales suponen la interfaz visual genérica de la aplicación y las diferentes páginas contenidas en ella por las que navegaremos, respectivamente. En el anexo F se muestran algunos ejemplos de dicha implementación con Vue y Vuetify.

Por un lado, en el directorio *layout* se encuentra el envoltorio por defecto (*default.vue*), que contiene el menú lateral desplegable (*v-navigation-drawer*) que el usuario utilizará para navegar por las principales páginas de la aplicación. También tiene una barra superior fija (*v-toolbar*) con el botón para desplegar el mencionado menú y el botón para navegar al acceso/registro o la información del usuario, según si no está autenticado ya o sí, respectivamente.

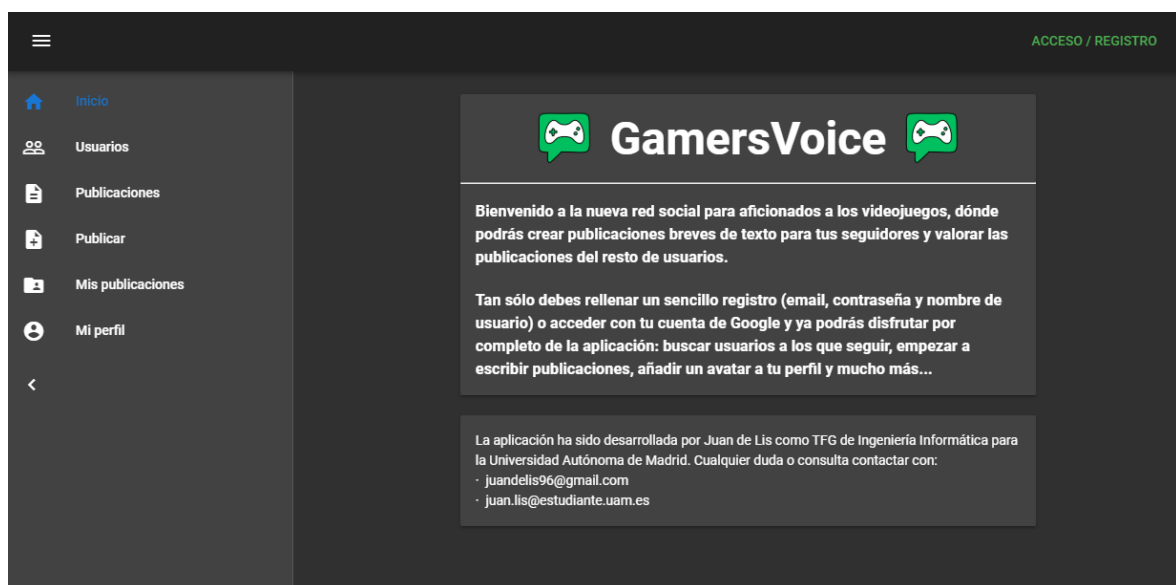


Figura 5-3: Vista inicial de la versión web

Por otro lado, en el directorio *pages* tendremos las páginas por las cuales navegará el usuario en la aplicación. En primer lugar, se ha implementado la página de inicio mostrada antes, que explica brevemente la aplicación y puede ser accedida tanto por usuarios autenticados como recién llegados. El resto de las páginas y funcionalidades requieren del registro y autenticación del usuario, que se podrá realizar en *login.vue*. En dicha página se configura y muestra FirebaseUI, la interfaz de acceso y registro que nos proporciona Firebase. La combinación de dicha interfaz y el agente de escucha de *user.js* se encargará por completo de la autenticación de usuarios en la aplicación.

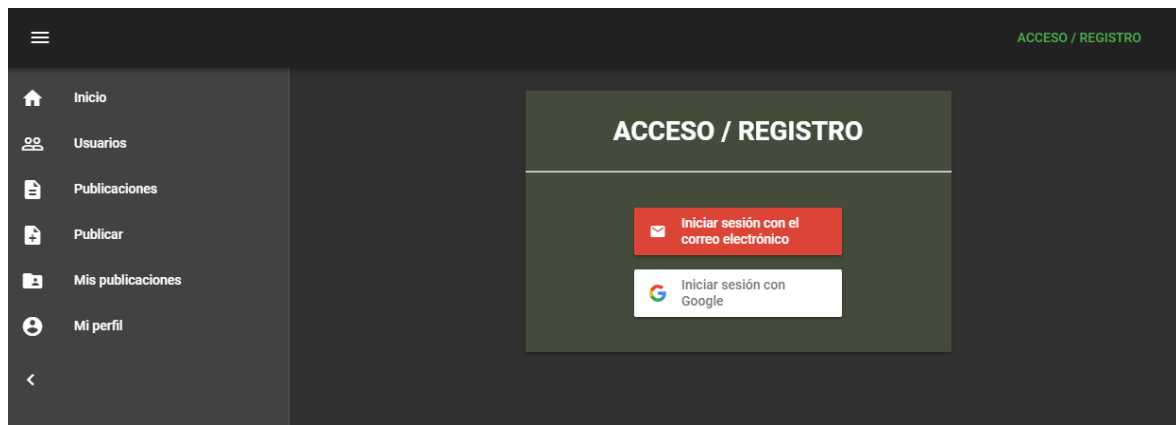


Figura 5-4: Vista del acceso/registro con FirebaseUI

Una vez el usuario se registre y acceda podrá navegar a su perfil (*\account*), dónde se le permite modificar o quitar la imagen de avatar, cambiar la contraseña e incluso borrar su cuenta por completo (*\account\delete*).

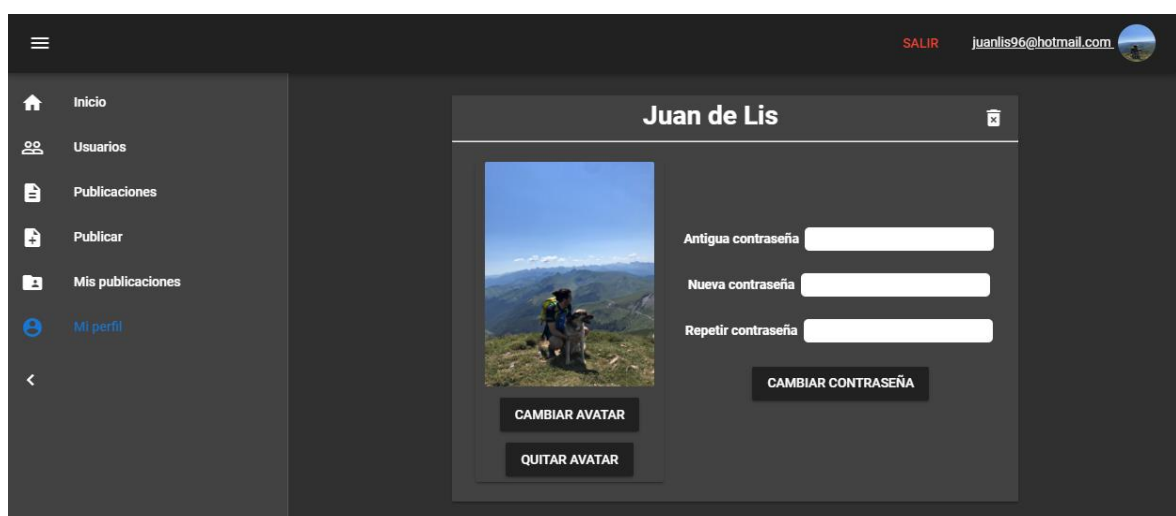


Figura 5-5: Vista del perfil del usuario autenticado

En lo que respecta al resto de usuarios de la aplicación, se ha implementado un buscador (`\users`) que permite filtrar por nombre de usuario y por la relación (los que sigues, los que no sigues o los que te siguen), así como visualizar el perfil del usuario deseado (`\users\<idUser>`). Esto último se realiza a través de rutas dinámicas, de forma que se implementa una página `\users_id` que obtiene el id introducido en la ruta como parámetro para mostrar al usuario indicado, reutilizando la misma página para todos.

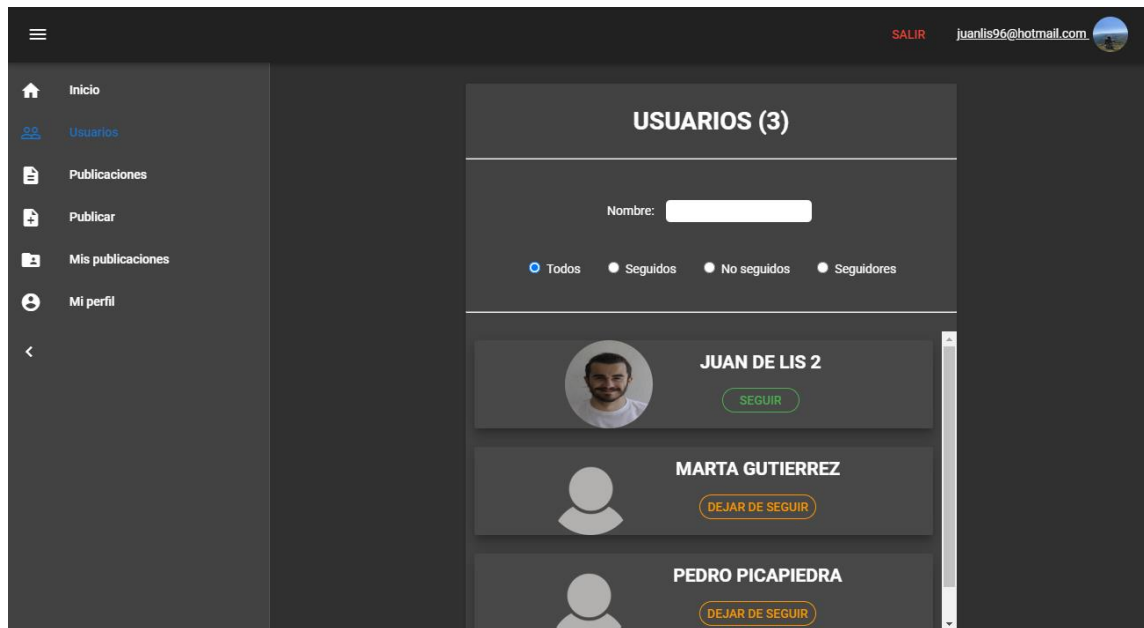


Figura 5-6: Vista del buscador de usuarios

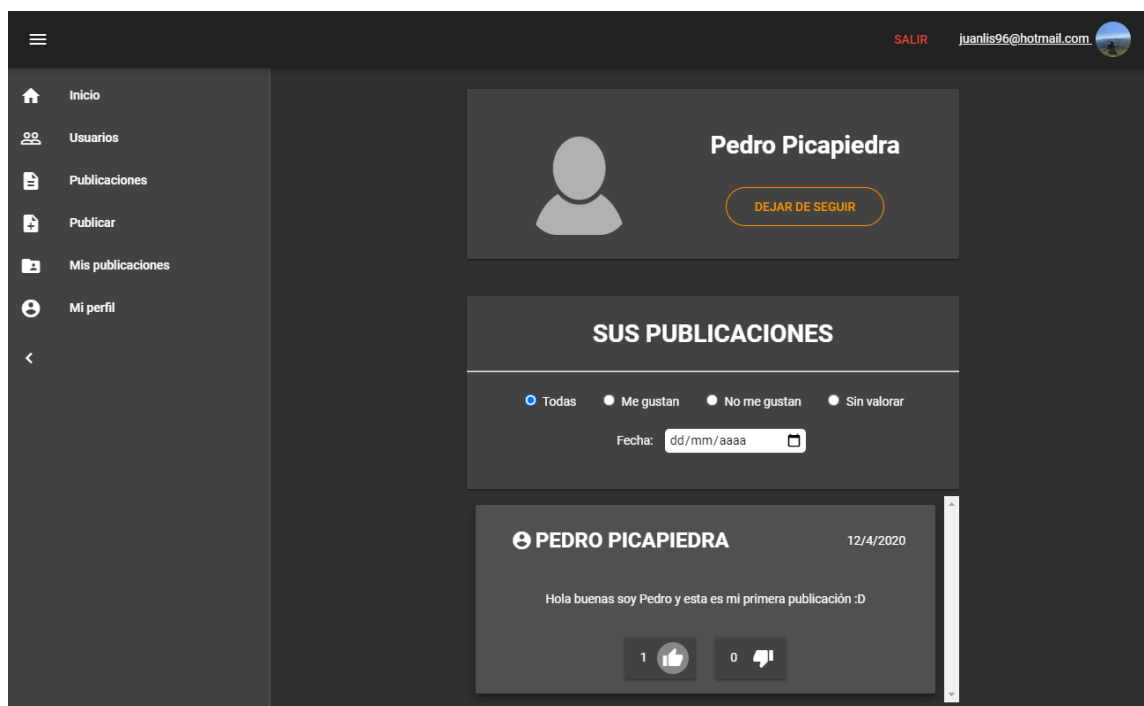


Figura 5-7: Vista del perfil de otro usuario

Respecto a las publicaciones, se ha implementado un buscador similar al de usuarios, que muestra las publicaciones de los usuarios seguidos y que permite filtrar por valoración o fecha. En una página similar el usuario será capaz de visualizar sus publicaciones, aunque esta vez el filtro será por texto de la publicación en lugar de por valoración (ya que el usuario no puede valorar sus propias publicaciones), además de poder borrar dichas publicaciones. En otra sencilla página el usuario podrá escribir nuevas publicaciones.

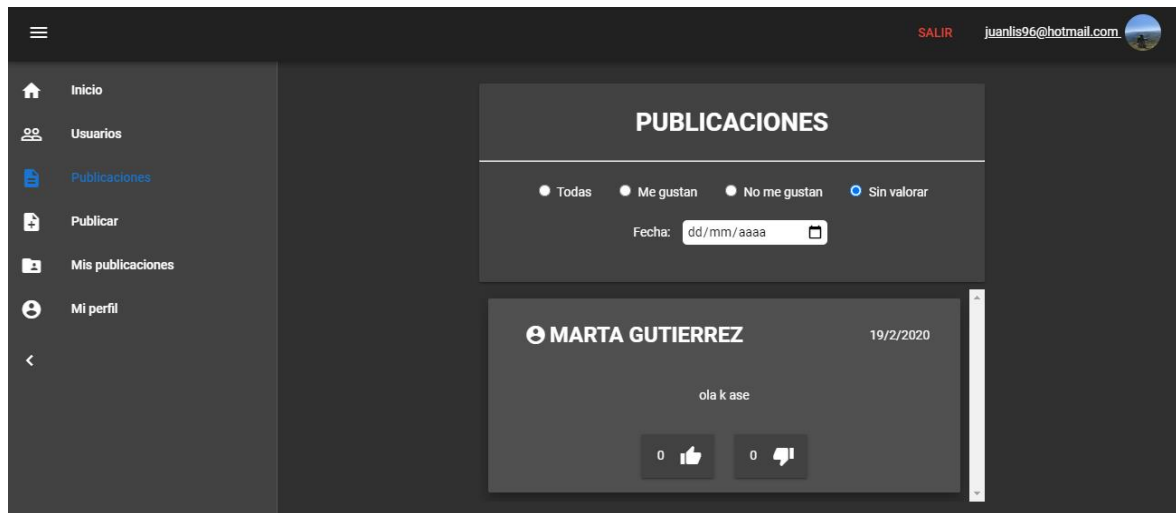


Figura 5-8: Vista de las publicaciones de usuarios seguidos

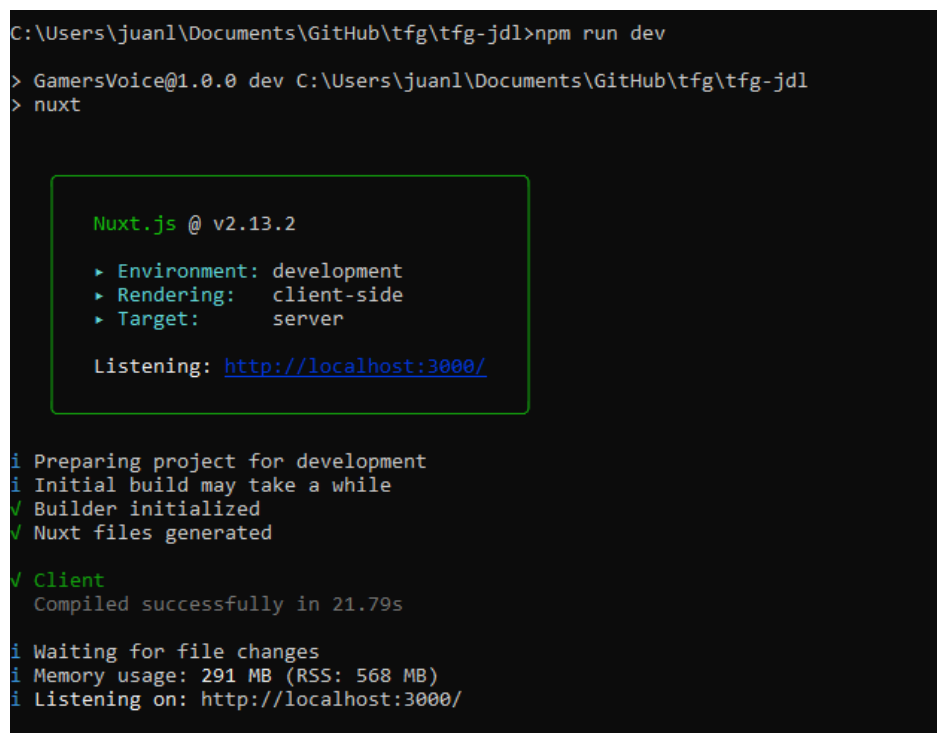
Debido a la instalación y configuración del módulo PWA en el proyecto de Nuxt, la interfaz de usuario se adapta perfectamente a dispositivos móviles pudiendo acceder a la aplicación web a través de cualquier navegador. Las capturas de pantalla de las vistas en la versión móvil se encuentran en el anexo G.

6. INTEGRACIÓN, PRUEBAS Y RESULTADOS

Durante el desarrollo de la aplicación es necesario el proceso de integración y pruebas de los diferentes componentes que se van implementando, de forma que después de cada iteración se compruebe que se han implementado los requisitos indicados y que los resultados son los esperados. Además, se debe comprobar la correcta integración de una iteración con las ya implementadas, de forma que se mantenga todo lo anterior correctamente y sin modificaciones de comportamiento no deseadas.

La metodología seguida ha consistido en realizar pruebas por cada módulo implementado, comprobando que se han cumplido los requisitos indicados (consultando también la consola de Firebase) y la correcta integración con el resto de los módulos.

Para llevar a cabo este proceso se ha aprovechado la capacidad de probar en local que ofrece Nuxt a través de su comando “nuxt” (npm run dev), que como ya se ha comentado compila y lanza la aplicación de forma local (en *localhost:3000*). De esta forma se pueden realizar las pruebas de la aplicación de forma rápida, ya que se lanza en modo desarrollo y se recarga a medida que realizamos modificaciones en el código, de forma que podemos ir comprobando los resultados de los cambios introducidos.



```
C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl>npm run dev

> GamersVoice@1.0.0 dev C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl
> nuxt

Nuxt.js @ v2.13.2
  > Environment: development
  > Rendering:   client-side
  > Target:      server

Listening: http://localhost:3000/

i Preparing project for development
i Initial build may take a while
✓ Builder initialized
✓ Nuxt files generated
✓ Client
  Compiled successfully in 21.79s
i Waiting for file changes
i Memory usage: 291 MB (RSS: 568 MB)
i Listening on: http://localhost:3000/
```

Figura 6-1: Lanzamiento de la aplicación en local

Una vez finalizadas las pruebas en local, se ejecuta el comando “nuxt build” para compilar y generar una versión estática de la aplicación en el directorio *\dist*, que será lo que se debe cargar en el servidor externo (Firebase Hosting).

Una vez generada la aplicación es necesario utilizar los comandos de Firebase para desplegarla en su servidor. El comando para desplegar todos los servicios de Firebase es “firebase deploy”, con el que se cargan las reglas del Storage (*storage.rules*) y Firestore (*firestore.rules*), las funciones de Cloud Functions (*\functions\index.js*) y el Hosting con la aplicación generada (*\dist*). El comando permite ignorar algún servicio (“*firebase deploy --except functions*”) o subir sólo los indicados (“*firebase deploy --only hosting*”), agilizando así las interacciones con el servidor.

En el anexo H se muestran ejemplos del uso de los comandos descritos.

Esta generación y carga de la aplicación ha resultado útil para la realización de las pruebas desde dispositivos móviles, comprobando una vez finalizadas las pruebas locales en el ordenador que la adaptación a dispositivos móviles es la deseada y también se siguen cumpliendo todos los requisitos planteados.

El resultado de todo este proyecto es una aplicación sencilla y funcional, que gracias a las tecnologías implementadas nos ofrece una gran escalabilidad del sistema. En primer lugar, los servicios de Firebase escalan automáticamente según las necesidades y el uso de la aplicación. Por otro lado, la generación estática del *frontend* se compatibiliza sin problemas con el posible aumento de usuarios, asegurando además que todos los usuarios utilizan la última versión implantada.

Para complementar la escalabilidad del sistema se utilizarán las herramientas de Firebase que permiten visualizar el uso de los diferentes servicios de nuestro *backend*. Del cual podemos ver capturas de pantalla de cómo se muestran dichos datos en el anexo I, aunque la aplicación está recientemente sacada en producción y los datos mostrados son casi nulos, nos da una idea de cómo visualizarlos e interpretarlos.

7. CONCLUSIONES Y TRABAJO FUTURO

7.1 Conclusiones

Este proyecto ha supuesto la creación de una red social para amantes de los videojuegos, con la intención de combinar el éxito de las redes sociales y el auge del sector de los videojuegos. La importancia de la realización de esta aplicación reside en el aprendizaje obtenido en cada una de las diferentes fases de esta.

El estudio de las diferentes tecnologías disponibles ha sido muy importante de cara al desarrollo del proyecto, ya que con él se ha podido ver la presencia de numerosos *frameworks* potentes para el desarrollo de aplicaciones web, y las diferentes opciones para llevarlo a cabo.

Por otro lado, ha sido muy importante la realización de un análisis exhaustivo, de cara a facilitar la implementación después del software, ya que se han podido extraer en detalle los requisitos de la aplicación y las necesidades de los futuros usuarios. Además, el diseño y las tecnologías utilizadas han resultado muy útiles para el desarrollo y las pruebas del código, gracias a la potencia y adaptabilidad del *framework* Nuxt y a la comodidad de usar un proveedor externo como Firebase para el *backend*.

Por todo ello, se considera que el resultado final ha sido satisfactorio, dando lugar a una aplicación sencilla pero potente que ha permitido explorar y aprender mucho sobre el desarrollo de aplicaciones web multiplataforma.

7.2 Trabajo futuro

Aparte de las necesarias correcciones de errores y optimizaciones que se deben realizar una vez que se ha lanzado la aplicación en producción, se han quedado algunas funcionalidades e ideas pendientes para futuras iteraciones. La principal de ellas es la de ampliar el contenido de las publicaciones actuales de sólo texto, permitiendo también la inclusión de contenido multimedia o las menciones a otros usuarios. Otra de las futuras implementaciones posibles sería la de incluir logros o premios en base a criterios de uso, éxito de publicaciones y de relaciones, por ejemplo, con la intención de aportar algo divertido y dinámico que pueda mantener motivados y activos a los usuarios.

REFERENCIAS

- [1] A. Ureña, A. Ferrari, D. Blanco y E. Valdecasa. “Las redes sociales en Internet”, Observatorio nacional de las telecomunicaciones y de la SI. Diciembre 2011.
- [2] D. Watts, “Seis Grados de Separación”, La ciencia de las redes en la era del acceso, Nueva York, Editorial Norton, Estados Unidos. 2003.
- [3] H. Harold Hutt, "Las redes sociales: una nueva herramienta de difusión", Revista reflexiones, 91(2). 2012.
- [4] AIMC, “Navegantes en la Red – Encuesta AIMC a usuarios de Internet”. Marzo 2020. http://download.aimc.es/aimc/Rub9aYt/naveg2019_principales_resultados.pdf
- [5] P. Alonqueo Boudon, L. Rehbein Felmer, “Usuarios habituales de videojuegos: una aproximación inicial”, Última década, 16(29):11-27. Diciembre 2008.
- [6] J. Díez, et al, “La diferencia sexual en el análisis de los videojuegos”, Instituto de la Mujer y Ministerio de Educación y Ciencia, Madrid. 2004.
- [7] D. Parra, et al., “Hábitos de uso de los videojuegos en España entre los mayores de 35 años” RLCS, Revista Latina de Comunicación Social, 64. 2009.
- [8] ISFE, “Games in society” Federación Europea de Software Interactivo, <https://www.isfe.eu/games-in-society/> Consultado el 5 de marzo de 2020.
- [9] A. Charland, B. Leroux, “Mobile Application Development: Web vs. Native”, Communications of the ACM, 54(5): 49-53, Mayo 2011.
- [10] R. Gibb, “What is a Web Application?”, Stackpath, Mayo 2016. <https://blog.stackpath.com/web-application/>
- [11] S. S. Tandel, A. Jamadar, “Impact of Progressive Web Apps on Web App Development”, International Journal of Innovative Research in Science, Engineering and Technology, 7(9), Septiembre 2018.
- [12] I. Melnik, “Single Page Application (SPA) vs Multi Page Application (MPA): Pros and Cons”, Abril 2020. <https://merehead.com/blog/single-page-application-vs-multi-page-application/>
- [13] B. Carter, “Grow your own Backend-as-a-Service (BaaS) platform”, Noviembre 2016. https://www.researchgate.net/publication/315497432_Grow_your_own_Backend-as-a-Service_BaaS_platform

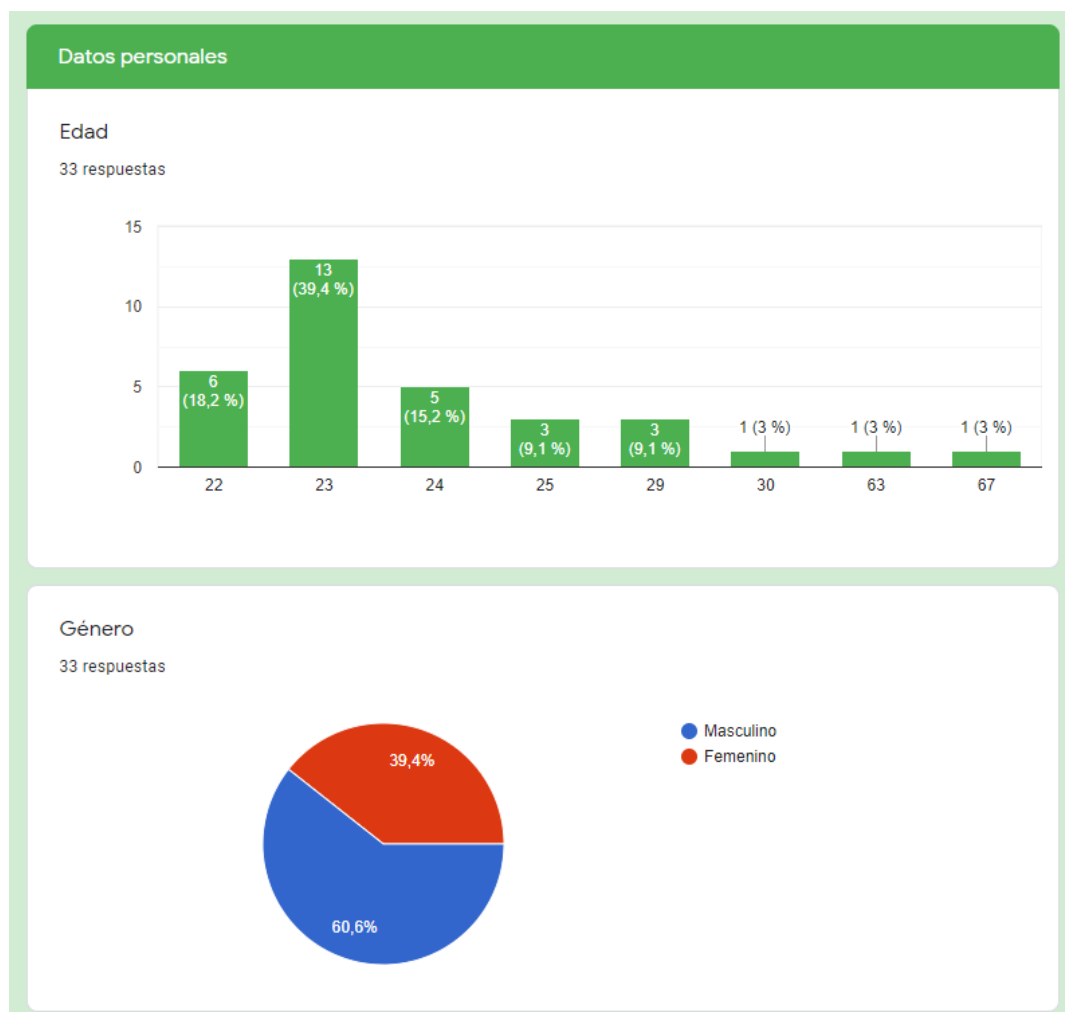
- [14] J. Gossman, “Introduction to Model/View/ViewModel pattern for building WPF apps”, Tales from the Smart Client, Microsoft, Agosto 2005.
<https://docs.microsoft.com/en-us/archive/blogs/johngossman/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps>
- [15] J. Smith, “Patterns - WPF Apps With The Model-View-ViewModel Design Pattern”, MSDN Magazine Issues and Downloads, 24 (2), febrero 2009.
<https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern>
- [16] G. DeMichillie, “Welcome Firebase to the Google Cloud Platform Team”, Google Cloud Platform Blog, Octubre 2014.
<https://cloudplatform.googleblog.com/2014/10/welcome-firebase-to-google-cloud-platform.html>
- [17] Firebase, “Documentación oficial de Firebase”, <https://firebase.google.com/docs> Consultado el 10 de abril de 2020
- [18] Nodejs.org, “About Node.js”, <https://nodejs.org/en/about/> Consultado el 10 de abril de 2020
- [19] Nuxtjs.org, “Guía de Nuxtjs”, <https://nuxtjs.org/guide> Consultado el 10 de abril de 2020
- [20] Vuejs.org, “Guía Vuejs”, <https://es.vuejs.org/v2/guide/> Consultado el 10 de abril de 2020
- [21] Vuetifyjs.com, “Why Vuetifyjs?”, <https://vuetifyjs.com/en/introduction/why-vuetify/> Consultado el 10 de abril de 2020
- [22] Vuexjs.org, “What is Vuex?”, <https://vuex.vuejs.org> Consultado el 10 de abril de 2020
- [23] B. Fisher, “Flux: A Unidirectional Data Flow Architecture for React Apps”. In Conference Applicative 2015, Association for Computing Machinery, Febrero 2015.
<https://doi.org/10.1145/2742580.2742818>

ANEXOS

A. Encuestas a futuros usuarios

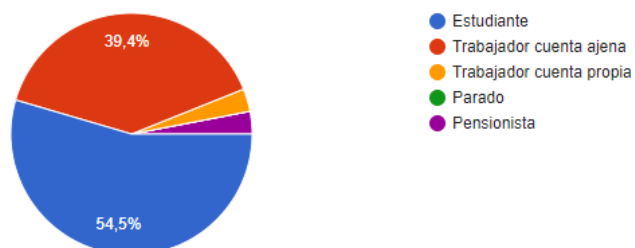
En el presente anexo podemos ver las preguntas, y sus correspondientes respuestas en forma de gráficas, de la encuesta que se ha realizado al comienzo del proyecto a personas cercanas que podían representar a los futuros usuarios de la aplicación, tanto por su interés en las redes sociales como por su interés en los videojuegos.

En primer lugar, se han realizado algunas preguntas de carácter más personal como la edad, género, ocupación e interés en los videojuegos con el objetivo de identificar a grandes rasgos el perfil de usuarios que han realizado la encuesta y su relevancia para el proyecto. Para finalizar, se ha preguntado a los usuarios sobre la aplicación, con la intención de obtener sus gustos y exigencias en una aplicación de tipo red social.

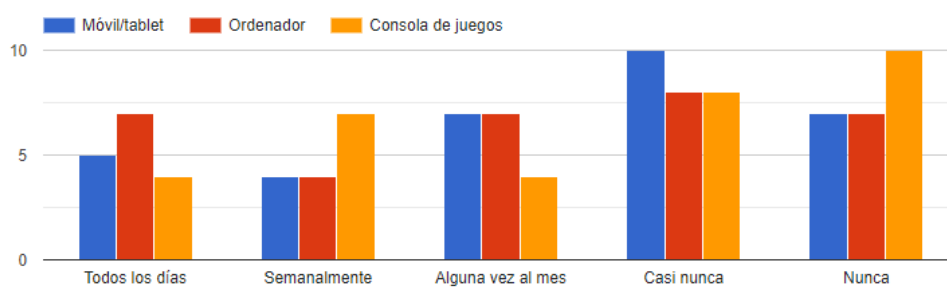


Situación laboral

33 respuestas



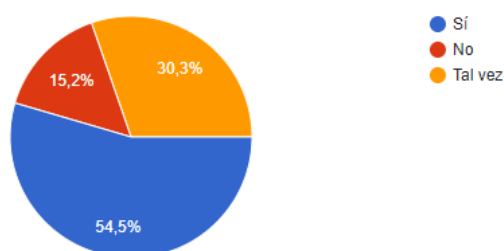
¿Con qué frecuencia juegas a videojuegos?



Preguntas sobre la aplicación

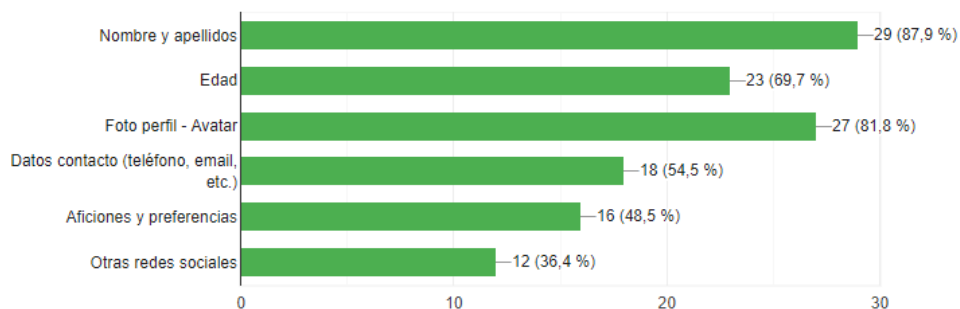
¿Usarías la aplicación si tuvieras que registrarte para utilizarla?

33 respuestas



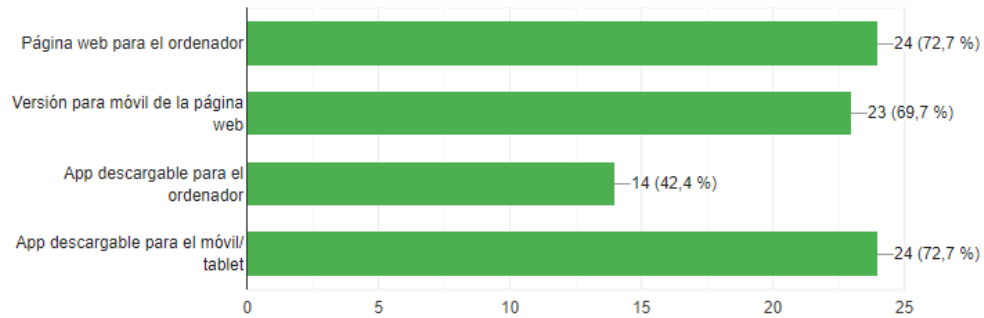
¿Qué información estarías dispuesto a aportar para tu perfil de usuario?

33 respuestas



¿Desde dónde quieres acceder?

33 respuestas



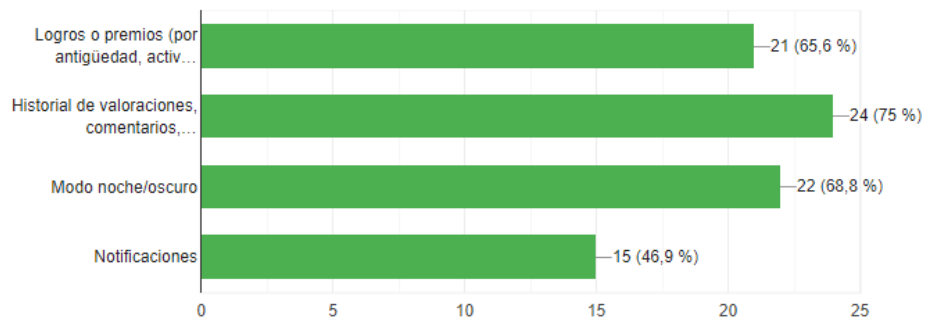
¿Qué características de red social te gustaría ver en la aplicación?

31 respuestas



¿Qué extras te parecen interesantes para la aplicación?

32 respuestas



B. Principales casos de uso detallados

Identificador: CU01	Nombre Caso de Uso: Autenticación en la aplicación
Autor: Juan de Lis	
Clases involucradas: Sistema, Usuario	
Resumen: Este caso de uso trata la autenticación de un usuario, previamente registrado, a la aplicación a través de su correo y contraseña del registro o de su cuenta de Google	
Precondiciones: El usuario deberá haberse registrado previamente	
Postcondiciones: El usuario accede correctamente y se le permite utilizar la aplicación	
Curso Básico de Eventos	
Usuario	Sistema
1. Introduce su correo 3. Introduce su contraseña	2. Comprueba que existe una cuenta con el correo introducido y solicita la contraseña 4. Comprueba que las credenciales son correctas y concede el acceso a la aplicación
Caminos Alternativos	
1.1. Se autentica con su cuenta de Google	1.2. Comprueba que su cuenta de Google es correcta y le concede acceso
2.1. Introduce un correo que no está registrado previamente 2.3. Completa el registro escogiendo un nombre de usuario y contraseña	2.2. Detecta que es un correo nuevo y solicita al usuario completar el registro 2.4. Registra al usuario y le concede acceso
3.1. Introduce un correo registrado pero una contraseña incorrecta	3.2. Advierte al usuario del error en las credenciales y permite introducirlas de nuevo

Identificador: CU02	Nombre Caso de Uso: Cambiar contraseña
Autor: Juan de Lis	
Clases involucradas: Sistema, Usuario	
Resumen: Este caso de uso trata el cambio de contraseña de un usuario previamente registrado y autenticado en la aplicación, que a través de su antigua contraseña puede establecer una nueva	
Precondiciones: El usuario deberá haberse registrado y accedido previamente en la aplicación	
Postcondiciones: La contraseña se modifica correctamente y podrá ser utilizada para autenticarse	
Curso Básico de Eventos	
Usuario	Sistema
1. Accede a la página del perfil del usuario 2. Introduce la contraseña antigua y la nueva	3. Comprueba que la contraseña antigua sea la correcta y la cambia por la nueva
Caminos Alternativos	
1.1. Introduce la antigua contraseña incorrectamente	1.2. Advierte de dicho error y permite al usuario introducir la contraseña de nuevo

Identificador: CU03	Nombre Caso de Uso: Cambiar avatar
Autor: Juan de Lis	
Clases involucradas: Sistema, Usuario	
Resumen: Este caso de uso trata el cambio de avatar de un usuario previamente registrado y autenticado en la aplicación, que puede seleccionar un fichero de tipo imagen (png o jpeg) almacenado en su equipo para ponerlo como avatar de su perfil	
Precondiciones: El usuario deberá haberse registrado y accedido previamente en la aplicación	
Postcondiciones: La imagen de perfil se modifica y se muestra correctamente	
Curso Básico de Eventos	
Usuario	Sistema
1. Accede a la página del perfil del usuario 2. Pulsa el botón de cambiar contraseña y selecciona un archivo de imagen en el selector de archivos que se muestra	3. Comprueba que el fichero introducido sea de tipo imagen y guarda dicho fichero como nueva imagen de perfil del usuario
Caminos Alternativos	
1.1. Introduce un fichero que no es de tipo imagen	1.2. Alerta por pantalla del error
2.1. El usuario pulsa el botón de quitar avatar	2.2. Se elimina el avatar actual y se deja el icono por defecto

Identificador: CU04	Nombre Caso de Uso: Crear publicación
Autor: Juan de Lis	
Clases involucradas: Sistema, Usuario, Publicación	
Resumen: Este caso de uso trata la creación de una publicación por un usuario previamente registrado y autenticado en la aplicación, mediante un breve texto que introducirá por pantalla	
Precondiciones: El usuario deberá haberse registrado y accedido previamente en la aplicación	
Postcondiciones: La publicación se crea y se muestra correctamente	
Curso Básico de Eventos	
Usuario	Sistema
1. Accede a la página de crear una publicación 2. Introduce el texto deseado y lo envía	3. Crea la publicación el texto introducido y la fecha de creación
Caminos Alternativos	
1.1. No introduce texto en la publicación	1.2. Advierte que no es posible crear una publicación vacía y que debe introducir texto

Identificador: CU05	Nombre Caso de Uso: Borrar publicación
Autor: Juan de Lis	
Clases involucradas: Sistema, Usuario, Publicación	
Resumen: Este caso de uso trata el borrado de una publicación previamente creada por el mismo usuario	
Precondiciones: El usuario deberá haberse creado previamente la publicación	
Postcondiciones: La publicación se borra y deja de visualizarse para el resto de los usuarios	
Curso Básico de Eventos	
Usuario	Sistema
1. Localiza la publicación propia 2. Pulsa el botón de borrar la publicación	3. Elimina la publicación de la base de datos
Caminos Alternativos	
1.1. Utiliza los filtros del buscador de publicaciones para encontrarla 1.3. Pulsa el botón de borrar la publicación	1.2. Filtra las publicaciones del usuario con los parámetros introducidos en el buscador 1.4. Elimina la publicación de la base de datos
2.1. El usuario pulsa el botón de quitar avatar	2.2. Se elimina el avatar actual y se deja el icono por defecto

Identificador: CU06	Nombre Caso de Uso: Valorar publicación
Autor: Juan de Lis	
Clases involucradas: Sistema, Usuario, Publicación	
Resumen: Este caso de uso trata la valoración de una publicación de otro usuario de la aplicación, pudiendo ser una valoración positiva o negativa	
Precondiciones: El usuario debe estar autenticado y que otros usuarios hayan creado publicaciones	
Postcondiciones: La valoración del usuario se añade a la publicación y se actualiza el contador	
Curso Básico de Eventos	
Usuario	Sistema
1. Busca en las publicaciones de otros usuarios seguidos la que quiere valorar 2. Pulsa el botón de valorar positivamente	3. Añade la valoración positiva del usuario a la publicación
Caminos Alternativos	
1.1. Utiliza los filtros del buscador de publicaciones para encontrarla 1.3. Pulsa el botón de valorar positivamente	1.2. Filtra las publicaciones de otros usuarios con los parámetros introducidos en el buscador 1.4. Añade la valoración positiva del usuario
2.1. Busca la publicación que quiere valorar 2.2. Pulsa el botón de valorar negativamente	2.3. Añade la valoración negativa del usuario a la publicación
2.1. Busca la publicación que quiere valorar 2.2. Pulsa el botón de valorar negativamente, habiendo valorado antes positivamente	2.3. Elimina la valoración positiva previa y añade la valoración negativa a la publicación
2.1. Busca la publicación que quiere valorar 2.2. Pulsa de nuevo el botón de valorar positivamente, ya valorado así previamente	2.3. Elimina la valoración positiva previa

Identificador: CU07	Nombre Caso de Uso: Seguir a otro usuario
Autor: Juan de Lis	
Clases involucradas: Sistema, Usuario	
Resumen: Este caso de uso trata la acción de crear o destruir una relación de seguimiento hacia otro usuario de la aplicación	
Precondiciones: El usuario deberá acceder previamente en la aplicación y existir otro usuario	
Postcondiciones: Se crea la relación hacia el otro usuario, cuyas publicaciones se empezarán a mostrar en la página correspondiente a publicaciones de usuarios seguidos	
Curso Básico de Eventos	
Usuario	Sistema
1. Localiza otro usuario de la aplicación 2. Pulsa el botón de empezar a seguir al usuario	3. Crea la relación hacia el otro usuario
Caminos Alternativos	
1.1. Utiliza los filtros del buscador de usuarios para encontrarlo 1.3. Pulsa el botón de empezar a seguir	1.2. Filtra los usuarios con los parámetros introducidos en el buscador 1.4. Crea la relación hacia el otro usuario
2.1. Localiza otro usuario que ya está siguiendo 2.2. Pulsa el botón de dejar de seguir al usuario	2.3. Elimina la relación hacia el otro usuario

Identificador: CU08	Nombre Caso de Uso: Ver publicaciones de otro usuario
Autor: Juan de Lis	
Clases involucradas: Sistema, Usuario, Publicación	
Resumen: Este caso de uso trata la visualización de publicaciones de otro usuario de la aplicación, el cual tendrá creadas previamente dichas publicaciones	
Precondiciones: El usuario estará autenticado y habrá otro usuario registrado con publicaciones ya creadas previamente	
Postcondiciones: Se visualizan las publicaciones del usuario seleccionado y se permite interactuar con ellas	
Curso Básico de Eventos	
Usuario	Sistema
1. Localiza otro usuario de la aplicación a través del buscador de usuarios 2. Pulsa en el usuario deseado y accede a su página personal	3. Muestra el perfil del usuario y sus publicaciones con el buscador correspondiente
Caminos Alternativos	
1.1. Localiza otro usuario de la aplicación a través de una publicación suya en la página de publicaciones 1.2. Pulsa en el usuario deseado y accede a su página personal	1.3. Muestra el perfil del usuario y sus publicaciones con el buscador correspondiente

C. Configuración y desarrollo de los servicios de Firebase

Ilustración 1: Dominios ofrecidos por Firebase Hosting

Dominios de gamersvoice	
Añadir dominio personalizado	
Dominio	Estado
gamersvoice.web.app	Predeterminado
gamersvoice.firebaseio.com	Predeterminado

Ilustración 2: Configuración del hosting en *firebase.json*

```
"hosting": {
  "site": "gamersvoice",
  "public": "dist",
  "ignore": [
    "firebase.json",
    "**/.*",
    "**/node_modules/**"
  ],
```

Ilustración 3: Configuración e instanciación de FirebaseUI en la vista de *login*

```
const uiConfig = {
  signInFlow: 'popup',
  credentialHelper: firebaseui.auth.CredentialHelper.NONE,
  // signInSuccessUrl: '<url-to-redirect-to-on-success>', //En Nuxt esto sería un problema.
  signInOptions: [
    // Leave the lines as is for the providers you want to offer your users.
    firebase.auth.EmailAuthProvider.PROVIDER_ID,
    firebase.auth.GoogleAuthProvider.PROVIDER_ID,
    // firebase.auth.PhoneAuthProvider.PROVIDER_ID,
    // firebaseui.auth.AnonymousAuthProvider.PROVIDER_ID
  ],
  callbacks: {
    signInSuccessWithAuthResult() {
      return false
    },
  },
}
const ui =
  firebaseui.auth.AuthUI.getInstance() || new firebaseui.auth.AuthUI(auth)
// The start method will wait until the DOM is loaded.
ui.start('#firebaseui-auth-container', uiConfig)
```

Ilustración 4: Configuración de proveedores permitidos en Firebase Authentication













Proveedores de inicio de sesión	
Proveedor	Estado
 Correo electrónico/contraseña	Habilitada
 Teléfono	Inhabilitado
 Google	Habilitada
 Play Juegos	Inhabilitado
 Game Center Beta	Inhabilitado
 Facebook	Inhabilitado
 Twitter	Inhabilitado
 GitHub	Inhabilitado
 Yahoo	Inhabilitado
 Microsoft	Inhabilitado
 Apple	Inhabilitado
 Anónimo	Inhabilitado

Ilustración 5: Reglas de seguridad en Cloud Storage

```
1 rules_version = "2";
2 service firebase.storage {
3   match /b/{bucket}/o {
4     match /profileImages/{file} {
5       allow read: if request.auth!=null;
6       allow write: if ((request.auth!=null) &&
7                        (request.auth.uid==file) &&
8                        (request.resource.contentType.matches('image/.*')));
9     }
10  }
11 }
```

Ilustración 6: Usuarios en Cloud Firestore

accounts	AToVGwlrD4dJVJsk6c1qlcvPGh23
+ Añadir documento	+ Iniciar colección
AToVGwlrD4dJVJsk6c1qlcvPGh23 >	+ Añadir campo
Ioon1C7ZEncw6nV63csc0DgkCoq1	email: "juanlis96@hotmail.com"
Nqs0Yg5kZOWGP1wZJQyj2wdPJkE2	image: "https://firebasestorage.googleapis.com/v0/b/tfg-jdl.appspot.com/o/profileImages%2FAToVGwlrD4dJVJsk6c1qlcvPGh23?alt=media&token=72480a1f-2b0a-4532-aa2a-94bff35a465e"
RhEhqTt7FePhlTuRF0gRU7e05w23	name: "Juan de Lis"

Ilustración 7: Reglas para los usuarios en Cloud Firestore

```

1 rules_version = "2";
2 service cloud.firestore {
3   match /databases/{database}/documents {
4     // True if the user is signed in
5     function signedIn() {
6       return (request.auth.uid != null);
7     }
8
9
10    match /accounts/{document} {
11      // True if the user is in its own document
12      function ownUserDocument() {
13        return (request.auth.uid == document);
14      }
15      // True if the user does not have a document yet
16      function newUser() {
17        return !(exists(/databases/{database}/documents/accounts/{request.auth.uid}));
18      }
19      // True if the update doesn't change email
20      function validOwnUserUpdate() {
21        return (request.resource.data.email == resource.data.email);
22      }
23
24      allow read: if signedIn();
25      allow create: if ownUserDocument() && newUser();
26      allow update: if (ownUserDocument() && validOwnUserUpdate());
27      allow delete: if ownUserDocument();
28    }
  }

```

Ilustración 8: Relaciones en Cloud Firestore

follows	0d75ZijwRem0lEHjKwVJ
+ Añadir documento	+ Iniciar colección
0d75ZijwRem0lEHjKwVJ >	+ Añadir campo
ZQCBukQJAMhChwNfvNJZ	date: 12 de junio de 2020, 1:15:03 UTC+2
tBkPwcxWPH3rf0XSwJk	dest: "RhEhqTt7FePhlTuRF0gRU7e05w23"
wcXosbH1AF605PvTAqSk	ori: "AToVGwlrD4dJVJsk6c1qlcvPGh23"

Ilustración 9: Reglas para las relaciones en Cloud Firestore

```
32 match /follows/{document} {  
33   allow read: if signedIn();  
34   allow create: if ((request.resource.data.ori == request.auth.uid) &&  
35                   (request.resource.data.dest != request.auth.uid));  
36   allow delete: if ((resource.data.ori == request.auth.uid) &&  
37                   (resource.data.dest != request.auth.uid));  
38 }
```

Ilustración 10: Publicaciones en Cloud Firestore




 posts	 bVOnJ8qleT7bypnldWnd
+ Añadir documento	+ Iniciar colección
6gVn4Z8eByiu0ByYRIvN	+ Añadir campo
7dCQiSIrmVVH3ldhRyht	body: "Hola buenas soy Pedro y esta es mi primera publicación :D"
bVOnJ8qleT7bypnldWnd >	creatorId: "RhEhqTt7FePhlTurFOgRU7eO5w23"
vvBxy9jyZTYUpqq5NkuU	creatorName: "Pedro Pérez"
xXUfr640bxYplgky7MxN	date: 12 de abril de 2020, 21:26:41 UTC+2
	▼ dislikes
	▼ likes
	 "AToVGwlrD4dJVJsk6c1qlcvPGh23"

Ilustración 11: Reglas para las publicaciones en Cloud Firestore

```
42 match /posts/{document} {
43   // True if the post is from another user
44   function otherUserDocument() {
45     return (request.auth.uid != null) &&
46       (resource.data.creatorId != request.auth.uid);
47   }
48   // True if the update only added/remove the uid in the likes/dislikes
49   function validOtherUserUpdate() {
50     return (request.resource.data.body == resource.data.body) &&
51       (request.resource.data.creatorId == resource.data.creatorId) &&
52       (request.resource.data.creatorName == resource.data.creatorName) &&
53       (request.resource.data.date == resource.data.date) &&
54       (
55         // Add uid to the likes
56         ((request.resource.data.likes.size() == resource.data.likes.size()+1) &&
57           !(request.auth.uid in resource.data.likes) &&
58           (request.resource.data.likes.hasAll(resource.data.likes.concat([request.auth.uid]))))
59         ||
60         // Remove uid from the likes
61         ((request.resource.data.likes.size() == resource.data.likes.size()-1) &&
62           (request.auth.uid in resource.data.likes) &&
63           (request.resource.data.likes.hasAll(resource.data.likes.removeAll([request.auth.uid]))))
64         ||
65         // Add uid to the dislikes
66         ((request.resource.data.dislikes.size() == resource.data.dislikes.size()+1) &&
67           !(request.auth.uid in resource.data.dislikes) &&
68           (request.resource.data.dislikes.hasAll(resource.data.dislikes.concat([request.auth.uid]))))
69         ||
70         // Remove uid from the dislikes
71         ((request.resource.data.dislikes.size() == resource.data.dislikes.size()-1) &&
72           (request.auth.uid in resource.data.dislikes) &&
73           (request.resource.data.dislikes.hasAll(resource.data.dislikes.removeAll([request.auth.uid]))
74         );
75     );
76   }
77   allow read: if signedIn();
78   allow create: if (signedIn() && (request.resource.data.creatorId == request.auth.uid));
79   allow update: if (otherUserDocument() && validOtherUserUpdate());
80   allow delete: if (signedIn() && (resource.data.creatorId == request.auth.uid));
81 }
82 }
```

Ilustración 12: Declaración de la función y variables necesarias en Cloud Functions

```
// The Cloud Functions for Firebase SDK to create Cloud Functions and setup triggers.
const functions = require('firebase-functions')

exports.deletedUser = functions.firestore
  .document('accounts/{userId}')
  .onDelete((snap, context) => {
    // The Firebase Admin SDK to access the Firebase Firestore Database.
    const admin = require('firebase-admin')
    //admin.initializeApp(FirebaseConfig);
    admin.initializeApp()
    const db = admin.firestore()
    const firestore = admin.firestore

    const userId = context.params.userId
    //const deletedData = snap.data()
```

Ilustración 13: Borrar el usuario en Firebase Authentication

```
// Borrar usuario de Firebase Authentication
admin
  .auth()
  .deleteUser(userId)
  .catch((error) => {
    console.log('Error deleting user. ', error)
  })
```

Ilustración 14: Borrar imagen del usuario en Cloud Storage

```
// Borrar imagen del storage
admin
  .storage()
  .bucket()
  .file('profileImages/' + userId)
  .delete()
  .catch((error) => {
    console.log('Error deleting profile image. ', error)
  })
```

Ilustración 15: Borrar publicaciones propias y valoraciones del resto en Cloud Firestore

```
// Borrar Publicaciones tuyas y borrar uid de likes/dislikes en el resto
db.collection('posts')
  .get()
  .then((snapshot) => {
    snapshot.forEach((doc) => {
      const postData = doc.data()
      // Si el post es del user, lo borramos
      if (postData.creatorId === userId) {
        db.collection('posts')
          .doc(doc.id)
          .delete()
          .catch((error) => {
            console.error('Error removing post document. ', error)
          })
      }
      // Si el post no es suyo, buscamos y borramos su uid en los likes y dislikes
      else {
        if (postData.likes.includes(userId)) {
          db.collection('posts')
            .doc(doc.id)
            .update({
              likes: firestore.FieldValue.arrayRemove(userId),
            })
        }
        if (postData.dislikes.includes(userId)) {
          db.collection('posts')
            .doc(doc.id)
            .update({
              dislikes: firestore.FieldValue.arrayRemove(userId),
            })
        }
      }
    })
  })
  .catch((err) => {
    console.log('Error getting posts documents. ', err)
  })
  return null
})
```

Ilustración 16: Borrar relaciones en Cloud Firestore

```
// Borrar relaciones Follows del user
db.collection('follows')
  .get()
  .then((snapshot) => {
    snapshot.forEach((doc) => {
      const followData = doc.data()
      // Borramos toda relacion que tenga al user como origen o destino
      if (followData.ori === userId || followData.dest === userId) {
        db.collection('follows')
          .doc(doc.id)
          .delete()
          .catch((error) => {
            console.error('Error removing follow document. ', error)
          })
      }
    })
    return null
  })
  .catch((error) => {
    console.log('Error getting follows documents. ', error)
  })
```


D. Capturas de pantalla del desarrollo del proyecto

Ilustración 17: Creación del proyecto en Nuxt

```
C:\Users\juanl\Documents>npx create-nuxt-app tfg-jdl

create-nuxt-app v3.1.0
  Generating Nuxt.js project in tfg-jdl
  Project name: GamersTalk
  Programming language: JavaScript
  Package manager: Npm
  UI framework: Vuetify.js
  Nuxt.js modules: Progressive Web App (PWA)
  Linting tools: ESLint
  Testing framework: None
  Rendering mode: Single Page App
  Deployment target: Static (Static/JAMStack hosting)
  Development tools: (Press <space> to select, <a> to toggle all, <i> to invert selection)
Warning: name can no longer contain capital letters
| Installing packages with npm
```

Ilustración 18: Resultado de la creación correcta del proyecto en Nuxt

```
> GamersTalk@1.0.0 lint:js C:\Users\juanl\Documents\tfg-jdl
> eslint --ext .js,.vue --ignore-path .gitignore . "--fix"

  Successfully created project GamersTalk

To get started:

  cd tfg-jdl
  npm run dev

To build & start for production:

  cd tfg-jdl
  npm run build
  npm run start
```

Ilustración 19: Fichero *config.js* con la configuración del proyecto de Firebase

```
1 // fichero con la configuración de firebase
2 export const FirebaseConfig = {
3   apiKey: 'AIzaSyCYmAGlXpXscmWDPz-zvmzEOmxFe_wr5gE',
4   authDomain: 'tfg-jdl.firebaseio.com',
5   databaseURL: 'https://tfg-jdl.firebaseio.com',
6   projectId: 'tfg-jdl',
7   storageBucket: 'tfg-jdl.appspot.com',
8   messagingSenderId: '409622511361',
9   appId: '1:409622511361:web:9135a534e4b61da7c36500',
10 }
```

Ilustración 20: Fichero *firenit.js* con la inicialización de Firebase y sus servicios

```
1  import * as firebase from 'firebase/app'
2  import { FirebaseConfig } from '~/services/config'
3  import 'firebase/auth'
4  import 'firebase/firestore'
5  import 'firebase/storage'
6
7  firebase.initializeApp(FirebaseConfig)
8
9  export const auth = firebase.auth()
10
11 export const db = firebase.firestore()
12 export const storage = firebase.storage()
13
14 function getCurrentUserPromise(auth) {
15   return new Promise((resolve, reject) => {
16     const unsubscribe = auth.onAuthStateChanged((user) => {
17       unsubscribe()
18       resolve(user)
19     }, reject)
20   })
21 }
22
23 export const getCurrentUser = async () => {
24   if (auth.currentUser) return auth.currentUser
25   const result = await getCurrentUserPromise(auth)
26   return result
27 }
28
29 export default firebase
```

Ilustración 21: Directorios y ficheros del proyecto de Nuxt con Firebase

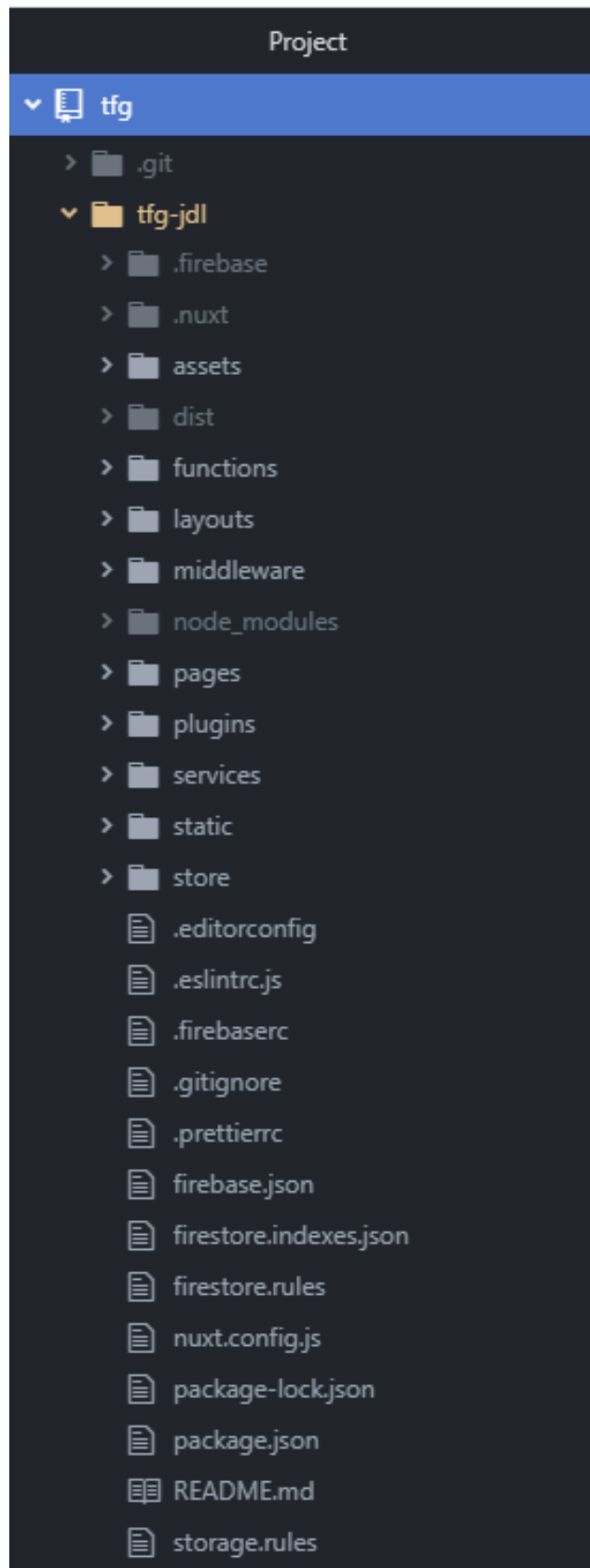


Ilustración 22: Método en `|assets|functions.js` para crear una publicación de Firestore

```
1  import { db } from '~/services/fireinit'
2
3  const functions = {
4    createPost(creatorId, creatorName, body, date) {
5      db.collection('posts')
6        .add({
7          creatorId,
8          creatorName,
9          body,
10         date,
11         dislikes: [],
12         likes: [],
13       })
14        .then(function () {
15          console.log('Post document successfully created!')
16        })
17        .catch(function (error) {
18          console.error('Error creating post document: ', error)
19        })
20    },
21  }
```

Ilustración 23: Método en `|assets|functions.js` para borrar una publicación de Firestore

```
22  deletePost(idPostToDelete) {
23    db.doc('posts/' + idPostToDelete)
24      .delete()
25      .then(function () {
26        console.log('Post document successfully deleted!')
27      })
28      .catch(function (error) {
29        console.error('Error removing post document: ', error)
30      })
31  },
```

Ilustración 24: Método en `|assets|functions.js` para borrar un usuario de Firestore

```
33  deleteUser(idUserToDelete) {
34    db.doc('accounts/' + idUserToDelete)
35      .delete()
36      .then(function () {
37        console.log('USER DOCUMENT SUCCESSFULLY DELETED!')
38      })
39      .catch(function (error) {
40        console.error('Error deleting user document: ', error)
41      })
42  },
43  }
```

Ilustración 25: Se comprueba que el usuario esté autenticado, en `|middleware|autenticado.js`

```
1 export default function ({ store, redirect, route }) {  
2   if (!store.getters['user/logged']) {  
3     store.commit('user/setAfterLogin', route.path)  
4     redirect('/login')  
5   }  
6 }
```

Ilustración 26: Registro de la librería Vuetify en Vue

```
1 import Vue from 'vue'  
2 import Vuetify from 'vuetify/lib'  
3 import colors from 'vuetify/es5/util/colors'  
4  
5 Vue.use(Vuetify, {  
6   theme: {  
7     primary: colors.blue.darken2,  
8     accent: colors.grey.darken3,  
9     secondary: colors.amber.darken3,  
10    info: colors.teal.lighten1,  
11    warning: colors.amber.base,  
12    error: colors.deepOrange.accent4,  
13    success: colors.green.accent3,  
14  },  
15 })
```

Ilustración 27: Logo de la aplicación, guardado en el directorio `static`

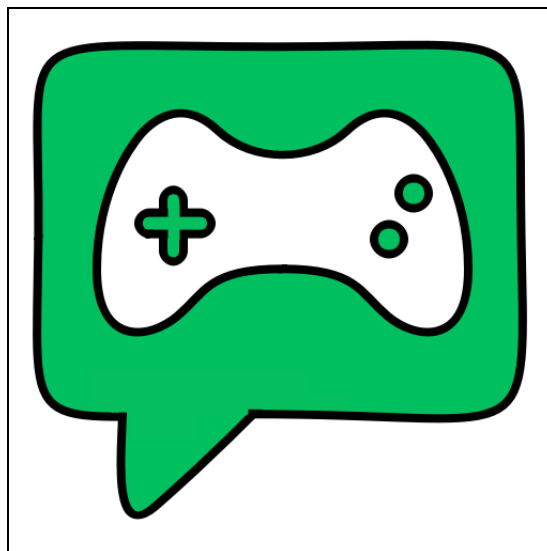


Ilustración 28: Fichero *nuxt.config* con la configuración de Nuxt

```
const VuetifyLoaderPlugin = require('vuetify-loader/lib/plugin')
const pkg = require('./package')
module.exports = {
  mode: 'spa',
  /*
   ** Headers of the page
   */
  head: {
    title: pkg.name,
    meta: [
      { charset: 'utf-8' },
      { name: 'viewport', content: 'width=device-width, initial-scale=1' },
      { hid: 'description', name: 'description', content: pkg.description }
    ],
    link: [
      {
        rel: 'stylesheet',
        href:
          'https://fonts.googleapis.com/css?family=Roboto:300,400,500,700|Material+Icons'
      }
    ]
  },
  /*
   ** Global CSS
   */
  css: ['~/assets/style/app.styl'],
  /*
   ** Plugins to load before mounting the App
   */
  plugins: ['@/plugins/vuetify'],

  /*
   ** Nuxt.js modules
   */
  modules: ['@nuxtjs/pwa'],
  /*
   ** Build configuration
   */
  build: {
    transpile: ['vuetify/lib'],
    plugins: [new VuetifyLoaderPlugin()],
    loaders: {
      stylus: {
        import: ['~/assets/style/variables.styl']
      }
    },
  },
  /*
   ** You can extend webpack config here
   */
  extend(config, ctx) {
    // Run ESLint on save
    if (ctx.isDev && ctx.isClient) {
      config.module.rules.push({
        enforce: 'pre',
        test: /\.?(js|vue)$/,
        loader: 'eslint-loader',
        exclude: /(node_modules)/,
        options: {
          fix: true
        }
      })
    }
  }
}
```

Ilustración 29: Fichero *package.json* con los datos, comandos y dependencias del proyecto

```
1  {
2    "name": "GamersTalk",
3    "version": "1.0.0",
4    "description": "Nuxt+Firebase web app for final degree proyect in Computer Science",
5    "author": "Juan de Lis",
6    "private": true,
7    "scripts": {
8      "dev": "nuxt",
9      "build": "nuxt build",
10     "start": "nuxt start",
11     "generate": "nuxt build && nuxt export",
12     "lint": "eslint --ext .js,.vue --ignore-path .gitignore .",
13     "precommit": "npm run lint"
14   },
15   "dependencies": {
16     "cross-env": "^7.0.2",
17     "firebase": "^7.15.5",
18     "firebaseui": "^4.5.1",
19     "material-icons": "^0.3.1",
20     "nuxt": "^2.13.2",
21     "vuetify": "^1.5.6",
22     "vuetify-loader": "^1.5.0"
23   },
24   "devDependencies": {
25     "@nuxtjs/eslint-config": "^3.0.0",
26     "@nuxtjs/pwa": "^3.0.0-beta.20",
27     "babel-eslint": "^10.1.0",
28     "core-js": "^2.5.7",
29     "eslint": "^7.3.1",
30     "eslint-config-prettier": "^6.11.0",
31     "eslint-config-standard": ">=14.1.1",
32     "eslint-loader": "^4.0.2",
33     "eslint-plugin-import": ">=2.22.0",
34     "eslint-plugin-jest": ">=23.17.1",
35     "eslint-plugin-node": ">=11.1.0",
36     "eslint-plugin-nuxt": ">=1.0.0",
37     "eslint-plugin-prettier": "3.1.4",
38     "eslint-plugin-promise": ">=4.2.1",
39     "eslint-plugin-standard": ">=4.0.1",
40     "eslint-plugin-vue": "^6.2.2",
41     "node-fetch": "^2.6.0",
42     "node-sass": "^4.14.1",
43     "nodemon": "^2.0.4",
44     "prettier": "2.0.5",
45     "sass-loader": "^8.0.2",
46     "stylus": "^0.54.7",
47     "stylus-loader": "^3.0.2"
48   }
49 }
```

E. Ejemplos de código de los módulos de Vuex

Ilustración 30: Imports, state y getter del módulo user.js

```
import { firestore } from 'firebase'
import { auth, getCurrentUser, db, storage } from '~/services/fireinit'

export const state = () => ({
  user: {
    uid: null, // no null si está logueado
    name: '',
    email: '',
    image: '',
    followers: [],
    following: [],
  },
  afterLogin: '/', // donde dirigirse una vez complete el login (por defecto el inicio)
  listeningAuth: false,
  unsubscribeUser: null, // guardará la función para dejar de escuchar cambios de User
  unsubscribeFollowers: null, // guardará la función para dejar de escuchar cambios en Followers
  unsubscribeFollowing: null, // guardará la función para dejar de escuchar cambios en Following
})

export const getters = {
  logged: (state, getters, rootState) => state.user.uid !== null,
}
```

Ilustración 31: Agente de escucha para seguidores en user.js

```
startListeningToFollowers({ state, commit }, userId) {
  // Nos ponemos en escucha de los documentos follow que tienen al user como destino (Followers)
  commit(
    'setUnsubscribeFollowers',
    db
      .collection('follows')
      .where('dest', '==', userId)
      .onSnapshot((followersSnapshot) => {
        // este código se ejecutará cuando se detecten cambios en los followers
        followersSnapshot.docChanges().forEach((change) => {
          const followData = change.doc.data()
          // Follower añadido
          if (change.type === 'added') commit('addFollower', followData.ori)
          // Follower borrado
          if (change.type === 'removed')
            commit('removeFollower', followData.ori)
        })
      })
  )
},

stopListeningToFollowers({ state }) {
  // Dejar de escuchar a cambios en los followers
  state.unsubscribeFollowers()
},
```

Ilustración 32: Agente de escucha para seguidos en *user.js*

```
startListeningToFollowing({ state, commit }, userId) {  
  // Nos ponemos en escucha de los documentos follow que tienen al user como destino (Following)  
  commit(  
    'setUnsubscribeFollowing',  
    db  
      .collection('follows')  
      .where('ori', '=', userId)  
      .onSnapshot((followingSnapshot) => {  
        // este código se ejecutará cuando se detecten cambios en los following  
        followingSnapshot.docChanges().forEach((change) => {  
          const followData = change.doc.data()  
          // Follower añadido  
          if (change.type === 'added') commit('addFollowing', followData.dest)  
          // Follower borrado  
          if (change.type === 'removed')  
            commit('removeFollowing', followData.dest)  
        })  
      })  
    )  
  },  
  
  stopListeningToFollowing({ state }) {  
    // Dejar de escuchar a cambios en los following  
    state.unsubscribeFollowing()  
  },  
}
```

Ilustración 33: Agente de escucha para el usuario autenticado en *user.js*

```
startListeningToUser({ state, commit, dispatch }, userDoc) {  
  // Nos ponemos en escucha del documento del usuario  
  commit(  
    'setUnsubscribeUser',  
    userDoc.onSnapshot((userDocSnapshot) => {  
      if (!userDocSnapshot.exists) {  
        // El user logged ha sido borrado  
        console.log('El doc del user logged ha sido borrado')  
        setTimeout(() => {  
          dispatch('logout')  
        }, 500)  
      } else {  
        // Borrar el viejo user  
        commit('clearUser')  
        // Guardar el nuevo user  
        const userData = userDocSnapshot.data()  
        commit('setUser', {  
          id: userDocSnapshot.id,  
          name: userData.name,  
          email: userData.email,  
          image: userData.image,  
        })  
      }  
    })  
  },  
  
  stopListeningToUser({ state, commit }) {  
    // Limpiar el user  
    commit('clearUser')  
    // Dejar de escuchar a cambios  
    state.unsubscribeUser()  
  },  
}
```

Ilustración 34: Agente de escucha para la autenticación en *user.js*

```
async initAuth({ state, commit, dispatch }) {
  if (!state.listeningAuth) {
    commit('setListeningAuth', true)
    auth.onAuthStateChanged((user) => {
      if (user) {
        // Login: buscamos el documento y empezamos a escuchar sus cambios
        const userDoc = db.doc('accounts/' + user.uid)
        userDoc
          .get()
          .then(function (doc) {
            if (doc.exists) {
              dispatch('startListeningToUser', userDoc)
              dispatch('startListeningToFollowers', user.uid)
              dispatch('startListeningToFollowing', user.uid)
            } else {
              // Si todavía no existe el documento del user logueado lo creamos
              userDoc
                .set({
                  email: user.email,
                  image: '/default-profile.png', // imagen por defecto, editable luego
                  name: user.displayName || '????',
                })
                .then(function () {
                  // Con el usuario logueado y su documento creado empezamos a escuchar
                  dispatch('startListeningToUser', userDoc)
                  dispatch('startListeningToFollowers', user.uid)
                  dispatch('startListeningToFollowing', user.uid)
                })
                .catch(function (error) {
                  console.log('Error creando el documento: ' + error)
                  user.delete().catch(function (error) {
                    console.log('Error borrando el usuario: ' + error)
                  })
                })
            }
          })
          .catch(function (error) {
            console.log('Error getting document: ', error)
          })
      } else if (state.user.uid) {
        // Si hay user en el state es que ha hecho logout
        // Dejamos de escuchar cambios
        dispatch('stopListeningToFollowing')
        dispatch('stopListeningToFollowers')
        dispatch('stopListeningToUser')
        // Limpiamos store
        commit('clearUnsubscribes')
        commit('clearUser')
      }
    })
    const user = await getCurrentUser() // Obtiene el usuario si no se cerró sesión
    const prevUid = state.user.uid
    const newUid = user ? user.uid : null
    if (prevUid !== newUid) commit('clearUser')
  }
},
```

Ilustración 35: Cierre de sesión en *user.js*

```
async logout({ commit, dispatch }) {  
  await auth.signOut()  
  this.$router.push('/')  
},
```

Ilustración 36: Comenzar a seguir a otro usuario, en *user.js*

```
follow({ state, commit, dispatch }, idUserToFollow) {  
  const userLoggedId = state.user.uid  
  if (userLoggedId) {  
    db.collection('follows').add({  
      date: firestore.Timestamp.now(),  
      dest: idUserToFollow,  
      ori: userLoggedId,  
    })  
  }  
},
```

Ilustración 37: Dejar de seguir a otro usuario, en *user.js*

```
unfollow({ state, commit, dispatch }, idUserToUnfollow) {  
  const userLoggedId = state.user.uid  
  if (userLoggedId) {  
    db.collection('follows')  
      .where('ori', '==', userLoggedId)  
      .where('dest', '==', idUserToUnfollow)  
      .get()  
      .then((querySnapshot) => {  
        querySnapshot.forEach((doc) => {  
          doc.ref.delete().catch(function (error) {  
            console.error('Error removing follow document: ', error)  
          })  
        })  
      })  
      .catch(function (error) {  
        console.log('Error getting follow document: ', error)  
      })  
  }  
},
```


Ilustración 38: Actualizar imagen de usuario de Storage y Firestore, en *user.js*

```
updateUserImage({ state }, newImage) {  
  const userLoggedInId = state.user.uid  
  if (userLoggedInId) {  
    // Actualizamos la imagen en storage y su url en el doc del user  
    storage  
      .ref('profileImages/' + userLoggedInId)  
      .put(newImage)  
      .then((snapshot) => {  
        snapshot.ref.getDownloadURL().then(  
          (foundURL) => {  
            db.collection('accounts')  
              .doc(userLoggedInId)  
              .update({ image: foundURL })  
          },  
          (error) => {  
            console.log('Error getting DownloadURL. ', error)  
          }  
        )  
      })  
      .catch(function (error) {  
        return alert('Error insertando nueva imagen en storage. ', error)  
      })  
  }  
},
```

Ilustración 39: Eliminar imagen de usuario de Storage y Firestore, en *user.js*

```
deleteUserImage({ state }) {  
  const userLoggedInId = state.user.uid  
  if (userLoggedInId) {  
    // No borramos si ya tiene la imagen por defecto  
    if (state.user.image === '/default-profile.png') {  
      console.log('No se puede borrar la imagen por defecto. ')  
    } else {  
      // Borramos imagen del storage  
      storage  
        .ref('profileImages/' + userLoggedInId)  
        .delete()  
        .then(function () {  
          // Volver a imagen por defecto  
          db.collection('accounts')  
            .doc(userLoggedInId)  
            .update({ image: '/default-profile.png' })  
        })  
        .catch(function (error) {  
          console.log('Error deleting image from storage. ', error)  
        })  
    }  
  }  
},
```

Ilustración 40: *Import y state en userToShow.js*

```
import { db } from '~/services/fireinit'

export const state = () => ({
  userToShow: {
    id: null, // no null si está logueado
    name: '',
    image: '',
  },
  unsubscribeUserToShow: null, // guardará la función para dejar de escuchar (se invocará en beforeDestroy)
})
```

Ilustración 41: Agente de escucha del usuario a mostrar en *userToShow.js*

```
startListeningToUserToShow({ state, commit }, idUserToShow) {
  // Nos ponemos en escucha del documento del usuario
  commit(
    'setUnsubscribeUserToShow',
    db.doc('accounts/' + idUserToShow).onSnapshot((userToShowDocSnapshot) => {
      if (!userToShowDocSnapshot.exists) {
        // El usuario no existe o ha sido borrado
        this.$router.push('/users')
      } else {
        // Borrar el viejo userToShow
        commit('clearUserToShow')
        // Guardar el nuevo userToShow
        const userToShowId = userToShowDocSnapshot.id
        const userToShowData = userToShowDocSnapshot.data()
        if (userToShowData) {
          commit('setUserToShow', {
            id: userToShowId,
            name: userToShowData.name,
            image: userToShowData.image,
          })
        } else {
          console.log('No data found for user ' + userToShowId)
        }
      }
    })
  )
},
```

Ilustración 42: Agente de escucha a usuarios en *users.js*

```
startListeningToUsers({ dispatch, commit, rootState }, payload) {  
  // Nos ponemos en escucha de la colleccion de accounts  
  commit(  
    'setUnsubscribeUsers',  
    db.collection('accounts').onSnapshot((usersSnapshot) => {  
      // Cargar Los nuevos users, modificar los cambiados y quitar los borrados  
      usersSnapshot.docChanges().forEach((change) => {  
        const userLogged = rootState.user.user  
        // Ignoramos cambios en el usuario loggeado (estos estarán en user.js)  
        if (change.doc.id !== userLogged.uid) {  
          const userData = change.doc.data()  
          if (  
            // Filtro nombre de usuario  
            userData.name.toLowerCase().includes(payload.name.toLowerCase()) &&  
            // Filtros relacion Follow  
            (payload.relation === 'all' ||  
              (payload.relation === 'followed' &&  
                userLogged.following.includes(change.doc.id)) ||  
              (payload.relation === 'notFollowed' &&  
                !userLogged.following.includes(change.doc.id)) ||  
              (payload.relation === 'follower' &&  
                userLogged.followers.includes(change.doc.id)))  
          ) {  
            // Users añadidos  
            if (change.type === 'added') {  
              commit('pushUser', {  
                id: change.doc.id,  
                name: userData.name,  
                image: userData.image,  
              })  
            }  
            // Users modificados  
            if (change.type === 'modified') {  
              commit('updateUser', {  
                id: change.doc.id,  
                name: userData.name,  
                image: userData.image,  
              })  
            }  
            // Users borrados  
            if (change.type === 'removed') {  
              commit('deleteUser', change.doc.id)  
            }  
          }  
        }  
      })  
    })  
  ),  
},
```

Ilustración 43: Agente de escucha de publicaciones en myposts.js

```
startListeningToPosts({ state, rootState, commit }, payload) {  
  // Nos ponemos en escucha de la colleccion de posts  
  commit(  
    'setUnsubscribeUsers',  
    db  
      .collection('posts')  
      .where('creatorId', '==', rootState.user.user.uid)  
      .orderBy('date', 'desc')  
      .onSnapshot((postsSnapshot) => {  
        // Cargar Los nuevos posts, modificar Los cambiados y quitar Los borrados  
        postsSnapshot.docChanges().forEach((change) => {  
          const postData = change.doc.data()  
          // Posts añadidos  
          if (change.type === 'added') {  
            if (  
              postData.body.toUpperCase().includes(payload.text.toUpperCase()) &&  
              (payload.date === ''  
                ? true  
                : postData.date.toDate().toISOString().split('T')[0] ===  
                  payload.date)  
            ) {  
              commit('pushPost', {  
                id: change.doc.id,  
                body: postData.body,  
                date: postData.date.toDate().toLocaleDateString('es-ES'),  
                likes: postData.likes,  
                dislikes: postData.dislikes,  
              })  
            }  
          }  
          // Posts modificados  
          if (change.type === 'modified') {  
            commit('updatePost', {  
              id: change.doc.id,  
              body: postData.body,  
              date: postData.date.toDate().toLocaleDateString('es-ES'),  
              likes: postData.likes,  
              dislikes: postData.dislikes,  
            })  
          }  
          // Posts borrados  
          if (change.type === 'removed') {  
            const index = state.posts.findIndex( (item) => item.id === change.doc.id )  
            commit('removePost', { index })  
          }  
        })  
      })  
  )  
},
```

Ilustración 44: Agente de escucha a publicaciones en *posts.js*

```
async startListeningToPosts({ state, rootState, commit }, payload) {
  let postsCollection = null
  // Obtenemos todos los posts o los del usuario indicado
  if (payload.creatorId === undefined) {
    postsCollection = await db.collection('posts').orderBy('date', 'desc')
  } else {
    commit('setCreatorId', payload.creatorId)
    postsCollection = await db.collection('posts')
      .where('creatorId', '==', payload.creatorId).orderBy('date', 'desc')
  }
  // Nos ponemos en escucha de la colección de posts
  commit(
    'setUnsubscribePosts',
    postsCollection.onSnapshot((postsSnapshot) => {
      postsSnapshot.docChanges().forEach((change) => {
        // Código que se ejecuta para cada cambio en postsCollection
        const postData = change.doc.data()
        const userLogged = rootState.user.user
        // Filtramos los posts obtenidos
        if (
          // Si no estamos en un usuario concreto, filtramos por usuarios seguidos
          (payload.creatorId === undefined
            ? userLogged.following.includes(postData.creatorId)
            : true) &&
          // Filtro de fecha (si la ha introducido)
          (payload.date === '' ? true
            : postData.date.toDate().toISOString().split('T')[0] === payload.date) &&
          // Filtro de valoración de los posts
          (payload.type === 'all' ||
            (payload.type === 'liked' &&
              postData.likes.includes(userLogged.uid)) ||
            (payload.type === 'disliked' &&
              postData.dislikes.includes(userLogged.uid)) ||
            (payload.type === 'notValued' &&
              !postData.likes.includes(userLogged.uid) &&
              !postData.dislikes.includes(userLogged.uid)))
        ) {
          // Posts añadidos
          if (change.type === 'added') {
            commit('pushPost', {
              id: change.doc.id,
              creatorName: postData.creatorName,
              creatorId: postData.creatorId,
              body: postData.body,
              date: postData.date.toDate().toLocaleDateString('es-ES'),
              uid: rootState.user.user.uid,
              likes: postData.likes,
              dislikes: postData.dislikes,
            })
          }
          // Posts modificados
          else if (change.type === 'modified') {
            commit('updatePost', {
              id: change.doc.id,
              uid: userLogged.uid,
              likes: postData.likes,
              dislikes: postData.dislikes,
            })
          }
          // Posts borrados
          else if (change.type === 'removed') {
            commit('removePost', change.doc.id)
          }
        }
      })
    })
  ),
},
```

Ilustración 45: Añadir valoración positiva a una publicación en *posts.js*

```
async like({ state, rootState, commit, dispatch }, payload) {
  const userLogged = rootState.user.user
  if (userLogged) {
    const docRef = await db.collection('posts').doc(payload.postId)
    if (payload.disliked) {
      // Remove user id from dislikes and add it to likes
      docRef.update({
        dislikes: firestore.FieldValue.arrayRemove(userLogged.uid),
        likes: firestore.FieldValue.arrayUnion(userLogged.uid),
      })
    } else {
      // Just add user id to likes of post
      docRef.update({
        likes: firestore.FieldValue.arrayUnion(userLogged.uid),
      })
    }
  }
},
```

Ilustración 46: Quitar valoración positiva a una publicación en *posts.js*

```
async unlike({ state, rootState, commit, dispatch }, idPostToLike) {
  const userLogged = rootState.user.user
  if (userLogged) {
    const docRef = await db.collection('posts').doc(idPostToLike)
    // Remove user id from likes of post
    docRef.update({
      likes: firestore.FieldValue.arrayRemove(userLogged.uid),
    })
  }
},
```

Ilustración 47: Quitar valoración negativa a una publicación en *posts.js*

```
async undislike({ state, rootState, commit, dispatch }, idPostToDislike) {
  const userLogged = rootState.user.user
  if (userLogged) {
    const docRef = await db.collection('posts').doc(idPostToDislike)
    // Remove user id from dislikes of post
    docRef.update({
      dislikes: firestore.FieldValue.arrayRemove(userLogged.uid),
    })
  }
},
```

Ilustración 48: Añadir valoración negativa a una publicación en *posts.js*

```
async dislike({ state, rootState, commit, dispatch }, payload) {
  const userLogged = rootState.user.user
  if (userLogged) {
    const docRef = await db.collection('posts').doc(payload.postId)
    if (payload.liked) {
      // Remove user id from likes and add it to dislikes
      docRef.update({
        likes: firestore.FieldValue.arrayRemove(userLogged.uid),
        dislikes: firestore.FieldValue.arrayUnion(userLogged.uid),
      })
    } else {
      // Just add user id to dislikes of post
      docRef.update({
        dislikes: firestore.FieldValue.arrayUnion(userLogged.uid),
      })
    }
  }
},
```

F. Ejemplos de código de la interfaz (Vue y Vuetify)

Ilustración 49: Ejemplo del *grid* de vuetify y su adaptabilidad a diferentes tamaños

Ilustración 50: Imports de la página de acceso/registro

Ilustración 51: Configuración e instanciación de FirebaseUI

```
const uiConfig = {
  signInFlow: 'popup',
  credentialHelper: firebaseui.auth.CredentialHelper.NONE,
  // signInSuccessUrl: '<url-to-redirect-to-on-success>', //En Nuxt esto sería un problema, ya que firebase-ui no usa vue-route
  signInOptions: [
    // Leave the lines as is for the providers you want to offer your users.
    firebase.auth.EmailAuthProvider.PROVIDER_ID,
    firebase.auth.GoogleAuthProvider.PROVIDER_ID,
    // firebase.auth.PhoneAuthProvider.PROVIDER_ID,
    // firebaseui.auth.AnonymousAuthProvider.PROVIDER_ID
  ],
  callbacks: {
    signInSuccessWithAuthResult() {
      return false
    },
  },
}
const ui =
  firebaseui.auth.AuthUI.getInstance() || new firebaseui.auth.AuthUI(auth)
// The start method will wait until the DOM is loaded.
ui.start('#firebaseui-auth-container', uiConfig)
```

Ilustración 52: Vista del buscador de publicaciones propias

```
<template>
  <v-layout justify-center>
    <v-flex text-xs-center xs12 sm10 md8 lg7 xl6>
      <div class="pb-2" color="#404540">
        <v-card>
          <br />
          <h1 align="center">MIS PUBLICACIONES</h1>
          <br />
          <hr />
          <br />
          <p>
            <label for="text"> Texto: </label> &nbsp;
            <input
              id="text"
              v-model="text"
              type="search"
              size="30"
              @input="searchPosts({ text: text, date: date })"
            />
          <br /><br />
            <label for="date"> Fecha: </label> &nbsp;
            <input
              id="date"
              v-model="date"
              type="date"
              @input="searchPosts({ text: text, date: date })"
            />
          </p>
        </v-card>
      </div>
    </v-flex>
  </v-layout>
```

[illegible]

Ilustración 54: Lógica de las publicaciones propias

```
<script>
import { mapState, mapActions } from 'vuex'
import functions from '~/assets/functions'

export default {
  data: () => ({
    text: '',
    date: '',
    offsetTopMyPosts: 0,
  }),

  middleware: 'autenticado',

  computed: {
    ...mapState('user', ['user']),
    ...mapState('myPosts', ['posts']),
  },

  mounted() {
    // Recorremos todos los padres acumulando sus distancias hasta el top
    this.offsetTopMyPosts = 0
    let element = document.getElementById('myposts')
    while (element) {
      this.offsetTopMyPosts +=
        element.offsetTop - element.scrollTop + element.clientTop
      element = element.offsetParent
    }
    // Empezamos a escuchar cambios en mis publicaciones
    this.startListeningToPosts({ text: this.text, date: this.date })
  },

  beforeDestroy() {
    this.stopListeningToPosts()
  },

  methods: {
    ...mapActions('myPosts', [
      'startListeningToPosts',
      'stopListeningToPosts',
      'searchPosts',
    ]),

    deletePostAux(idPostToDelete) {
      functions.deletePost(idPostToDelete)
    },
  },
}
</script>
```

G. Capturas de pantalla de la versión móvil

Ilustración 55: Vista de la página de inicio

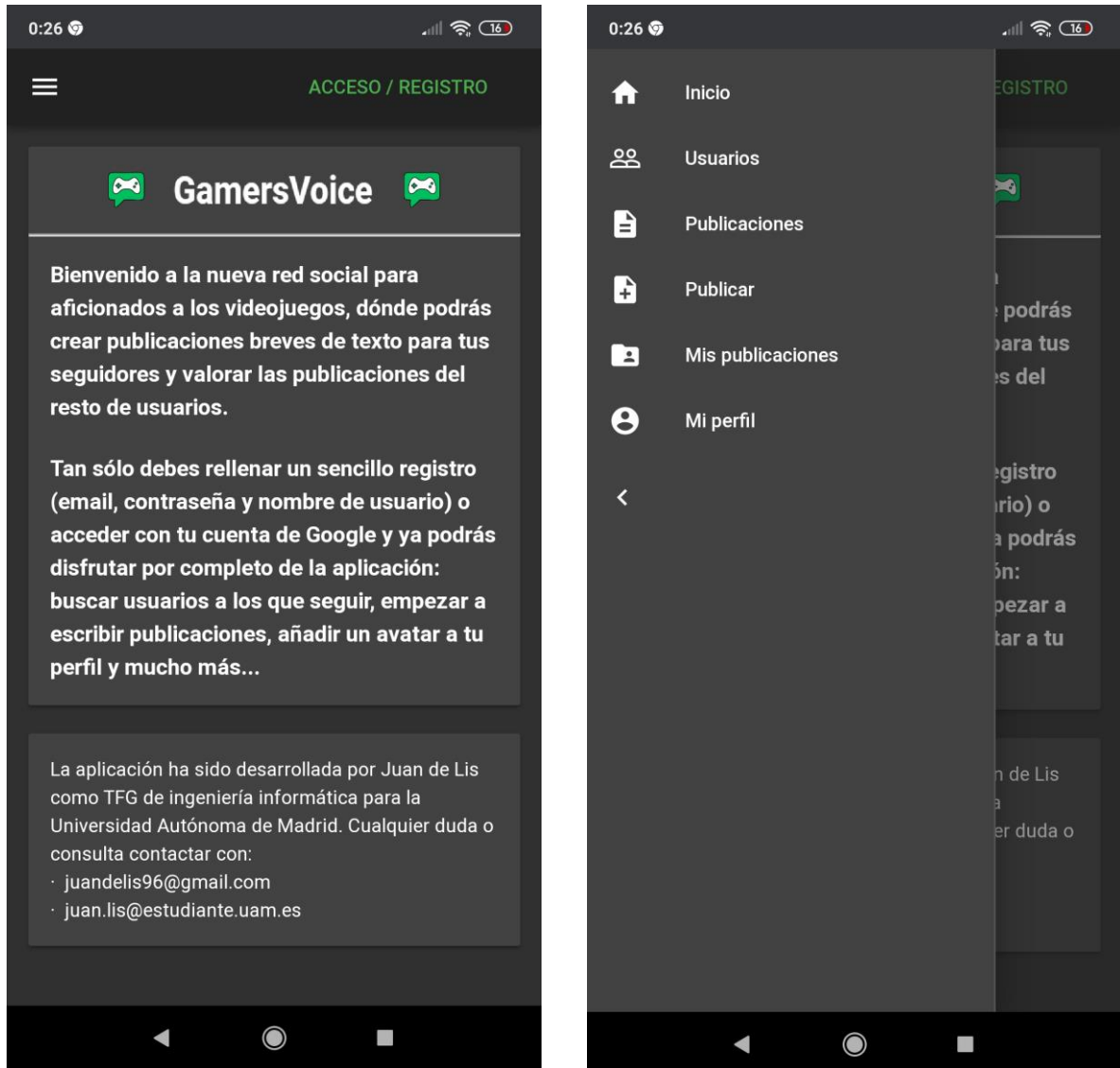


Ilustración 56: Vista de la autenticación del usuario

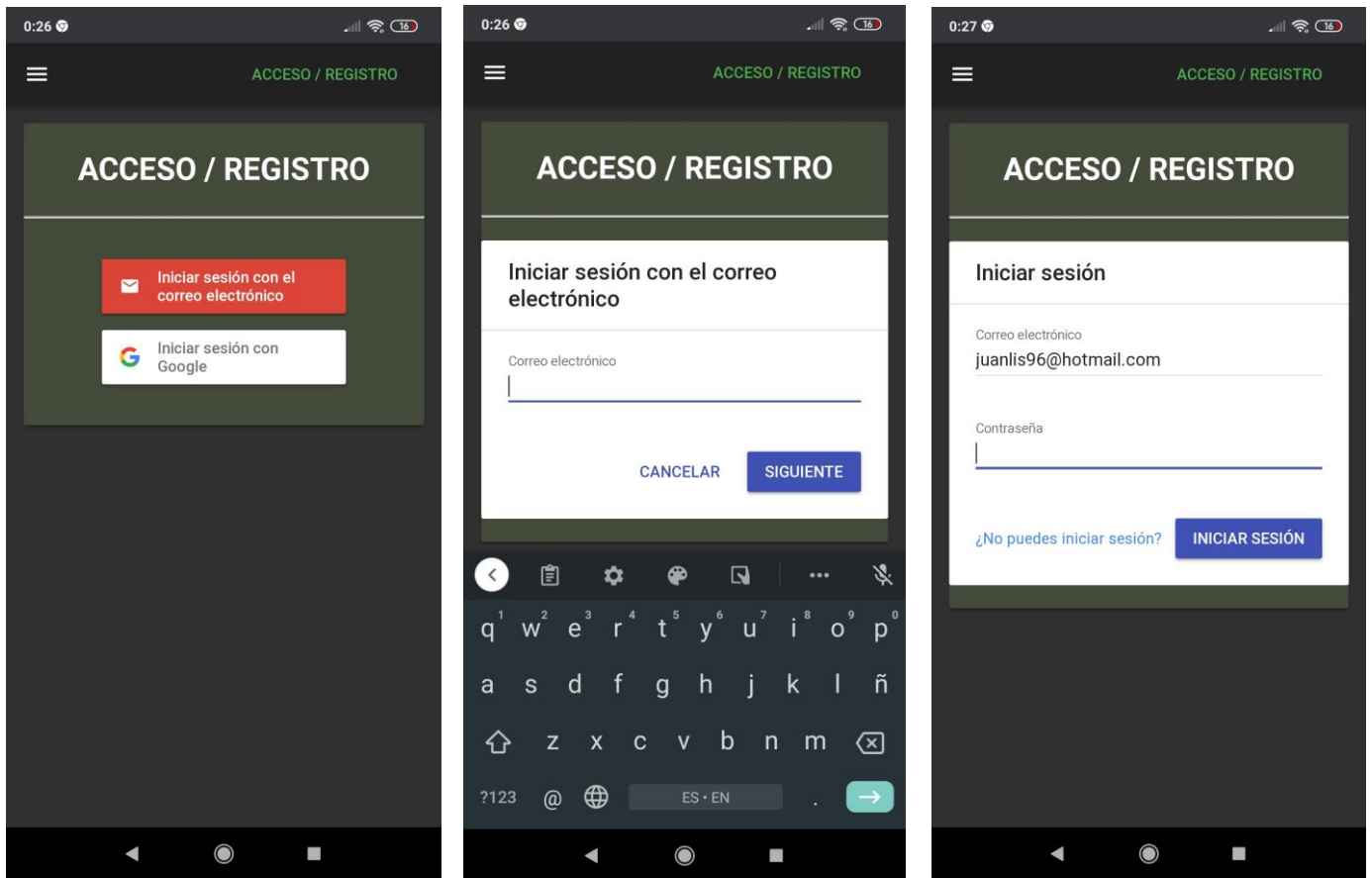


Ilustración 57: Vista del perfil del usuario autenticado

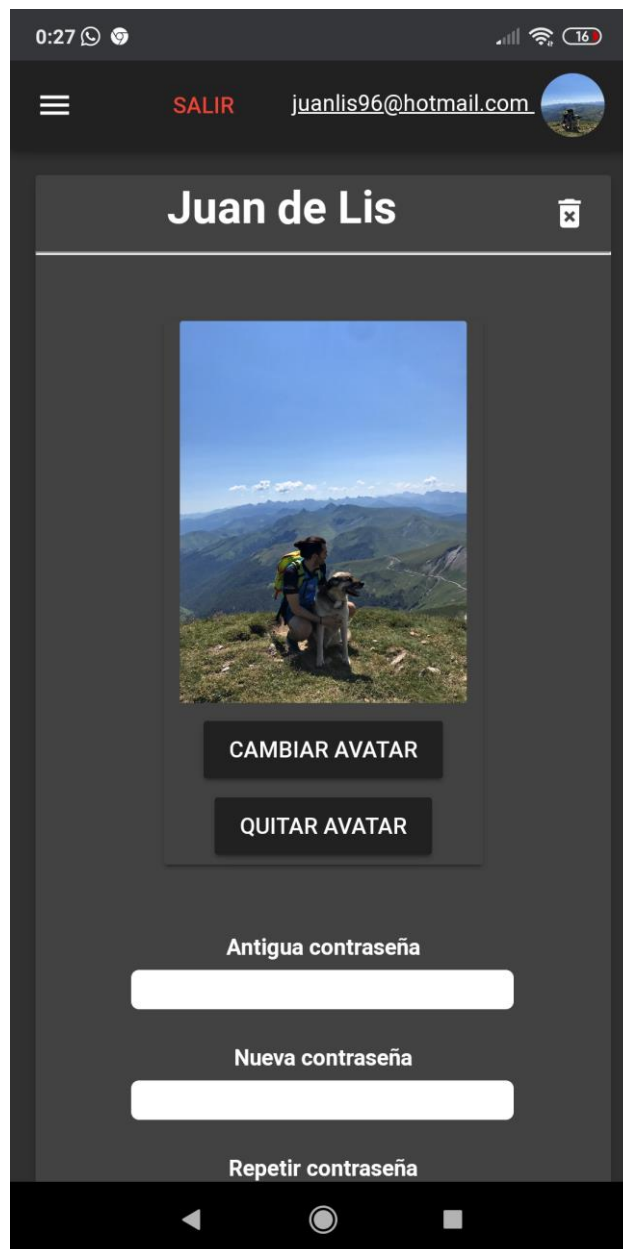


Ilustración 58: Vista del buscador y perfil de otros usuarios

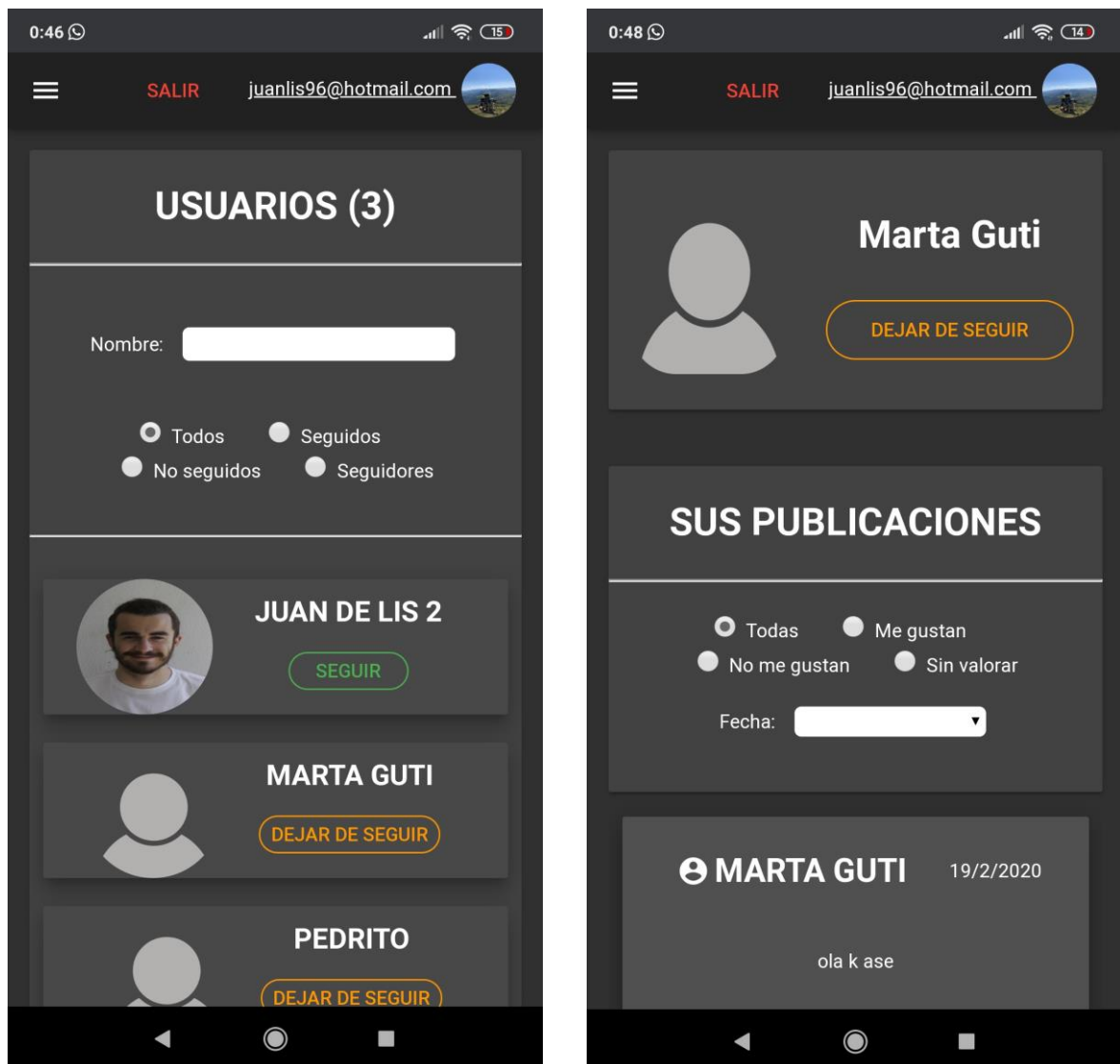
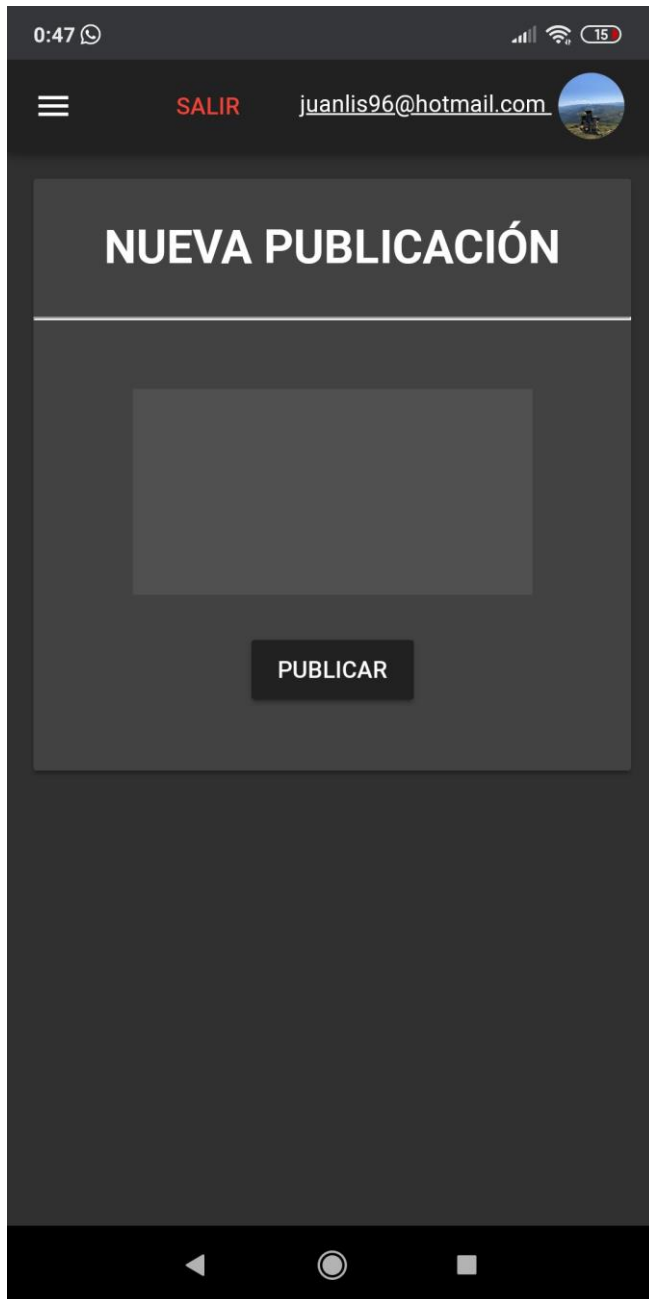


Ilustración 59: Vista del buscador de publicaciones de usuarios seguidos



Ilustración 60: Vista de la creación y buscador de publicaciones propias



H. Generación y despliegue de la aplicación

Ilustración 61: Generación del código estático con “nuxt build”

```
C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl>npm run build

> GamersVoice@1.0.0 build C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl
> nuxt build

i Production build
i Bundling only for client side
i Target: static
✓ Builder initialized
✓ Nuxt files generated

✓ Client
  Compiled successfully in 24.92s

Hash: 6fa15feed27c163a2427
Version: webpack 4.43.0
Time: 24925ms
Built at: 2020-07-08 21:12:33

```

Asset	Size	Chunks		Chunk Names
../server/client.manifest.json	13.8 KiB		[emitted]	
14.32bfffd5.js	4.3 KiB	14	[emitted] [immutable]	
LICENSES	4.39 KiB		[emitted]	
app.ed7fda4.js	133 KiB	0	[emitted] [immutable]	app
commons.app.816909d.js	165 KiB	1	[emitted] [immutable]	commons.app
icons/icon_120.92cfbe.png	6.65 KiB		[emitted]	
icons/icon_144.92cfbe.png	8.31 KiB		[emitted]	
icons/icon_152.92cfbe.png	8.88 KiB		[emitted]	
icons/icon_192.92cfbe.png	11.6 KiB		[emitted]	
icons/icon_384.92cfbe.png	26.4 KiB		[emitted]	
icons/icon_512.92cfbe.png	28.9 KiB		[emitted]	
icons/icon_64.92cfbe.png	3.05 KiB		[emitted]	
manifest.38cd30ff.json	816 bytes		[emitted]	
pages/account/delete.f27ff1f.js	6.58 KiB	2	[emitted] [immutable]	pages/account/delete
pages/account/index.43d27af.js	9.34 KiB	3	[emitted] [immutable]	pages/account/index
pages/index.19df8de.js	12.4 KiB	4	[emitted] [immutable]	pages/index
pages/login.f3924a9.js	2.02 KiB	5	[emitted] [immutable]	pages/login
pages/posts/create.fbf3cc2.js	7.77 KiB	6	[emitted] [immutable]	pages/posts/create
pages/posts/index.7df01f4.js	15.4 KiB	7	[emitted] [immutable]	pages/posts/index
pages/posts/myposts.34eed65.js	15.1 KiB	8	[emitted] [immutable]	pages/posts/myposts
pages/users/[_]id.abbfa91.js	16.5 KiB	9	[emitted] [immutable]	pages/users/[_]id
pages/users/index.a92ade8.js	9.71 KiB	10	[emitted] [immutable]	pages/users/index
runtime.5459894.js	2.64 KiB	11	[emitted] [immutable]	runtime
vendors.app.692d4d3.js	900 KiB	12	[emitted] [immutable] [big]	vendors.app
vendors.pages/login.0340025.js	279 KiB	13	[emitted] [immutable] [big]	vendors.pages/login
+ 1 hidden asset				

```
Entrypoint app [big] = runtime.5459894.js commons.app.816909d.js vendors.app.692d4d3.js app.ed7fda4.js

i Generating output directory: dist/
i Generating pages
✓ Generated route "/"account"
✓ Generated route "/"login"
✓ Generated route "/"posts"
✓ Generated route "/"users"
✓ Generated route "/"account/delete"
✓ Generated route "/"posts/create"
✓ Generated route "/"posts/myposts"
✓ Generated route "/"
✓ Client-side fallback created: 200.html

C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl>
```

Ilustración 62: Despliegue del servidor de Firebase, con todos sus productos

```
C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl>firebase deploy

=== Deploying to 'tf-g-jdl'...

i deploying storage, firestore, functions, hosting
Running command: npm --prefix "$RESOURCE_DIR" run lint -- --fix

> functions@ lint C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl\functions
> eslint . "--fix"

C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl\functions\index.js
  51:13  warning  Avoid nesting promises  promise/no-nesting
  90:13  warning  Avoid nesting promises  promise/no-nesting

ⓘ 2 problems (0 errors, 2 warnings)

+ functions: Finished running predeploy script.
i firebase.storage: checking storage.rules for compilation errors...
+ firebase.storage: rules file storage.rules compiled successfully
i firestore: reading indexes from firestore.indexes.json...
i cloud.firestore: checking firestore.rules for compilation errors...
+ cloud.firestore: rules file firestore.rules compiled successfully
i functions: ensuring required API cloudfunctions.googleapis.com is enabled...
+ functions: required API cloudfunctions.googleapis.com is enabled
i storage: latest version of storage.rules already up to date, skipping upload...
+ firestore: deployed indexes in firestore.indexes.json successfully
i firestore: latest version of firestore.rules already up to date, skipping upload...
i functions: preparing functions directory for uploading...
i functions: packaged functions (41.46 KB) for uploading
+ functions: functions folder uploaded successfully
i hosting[gamersvoice]: beginning deploy...
i hosting[gamersvoice]: found 38 files in dist
+ hosting[gamersvoice]: file upload complete
+ storage: released rules storage.rules to firebase.storage
+ firestore: released rules firestore.rules to cloud.firestore
i functions: updating Node.js 8 function deletedUser(us-central1)...
+ functions[deletedUser(us-central1)]: Successful update operation.
i hosting[gamersvoice]: finalizing version...
+ hosting[gamersvoice]: version finalized
i hosting[gamersvoice]: releasing new version...
+ hosting[gamersvoice]: release complete

+ Deploy complete!

Project Console: https://console.firebase.google.com/project/tfg-jdl/overview
Hosting URL: https://gamersvoice.web.app

C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl>
```

Ilustración 63: Despliegue de Firebase exceptuando Cloud Functions y Storage

```
C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl>firebase deploy --except functions,storage

=== Deploying to 'tf-g-jdl'...

i  deploying firestore, hosting
i  firestore: reading indexes from firestore.indexes.json...
i  cloud.firestore: checking firestore.rules for compilation errors...
+  cloud.firestore: rules file firestore.rules compiled successfully
+  firestore: deployed indexes in firestore.indexes.json successfully
i  firestore: latest version of firestore.rules already up to date, skipping upload...
i  hosting[gamersvoice]: beginning deploy...
i  hosting[gamersvoice]: found 38 files in dist
+  hosting[gamersvoice]: file upload complete
+  firestore: released rules firestore.rules to cloud.firestore
i  hosting[gamersvoice]: finalizing version...
+  hosting[gamersvoice]: version finalized
i  hosting[gamersvoice]: releasing new version...
+  hosting[gamersvoice]: release complete

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/tfg-jdl/overview
Hosting URL: https://gamersvoice.web.app

C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl>_
```

Ilustración 64: Despliegue de Firebase, pero sólo el Hosting

```
C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl>firebase deploy --only hosting

=== Deploying to 'tf-g-jdl'...

i  deploying hosting
i  hosting[gamersvoice]: beginning deploy...
i  hosting[gamersvoice]: found 38 files in dist
+  hosting[gamersvoice]: file upload complete
i  hosting[gamersvoice]: finalizing version...
+  hosting[gamersvoice]: version finalized
i  hosting[gamersvoice]: releasing new version...
+  hosting[gamersvoice]: release complete

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/tfg-jdl/overview
Hosting URL: https://gamersvoice.web.app

C:\Users\juanl\Documents\GitHub\tfg\tfg-jdl>_
```

I. Herramienta de uso y facturación de Firebase

Ilustración 65: Visión general del uso y facturación en la consola de Firebase

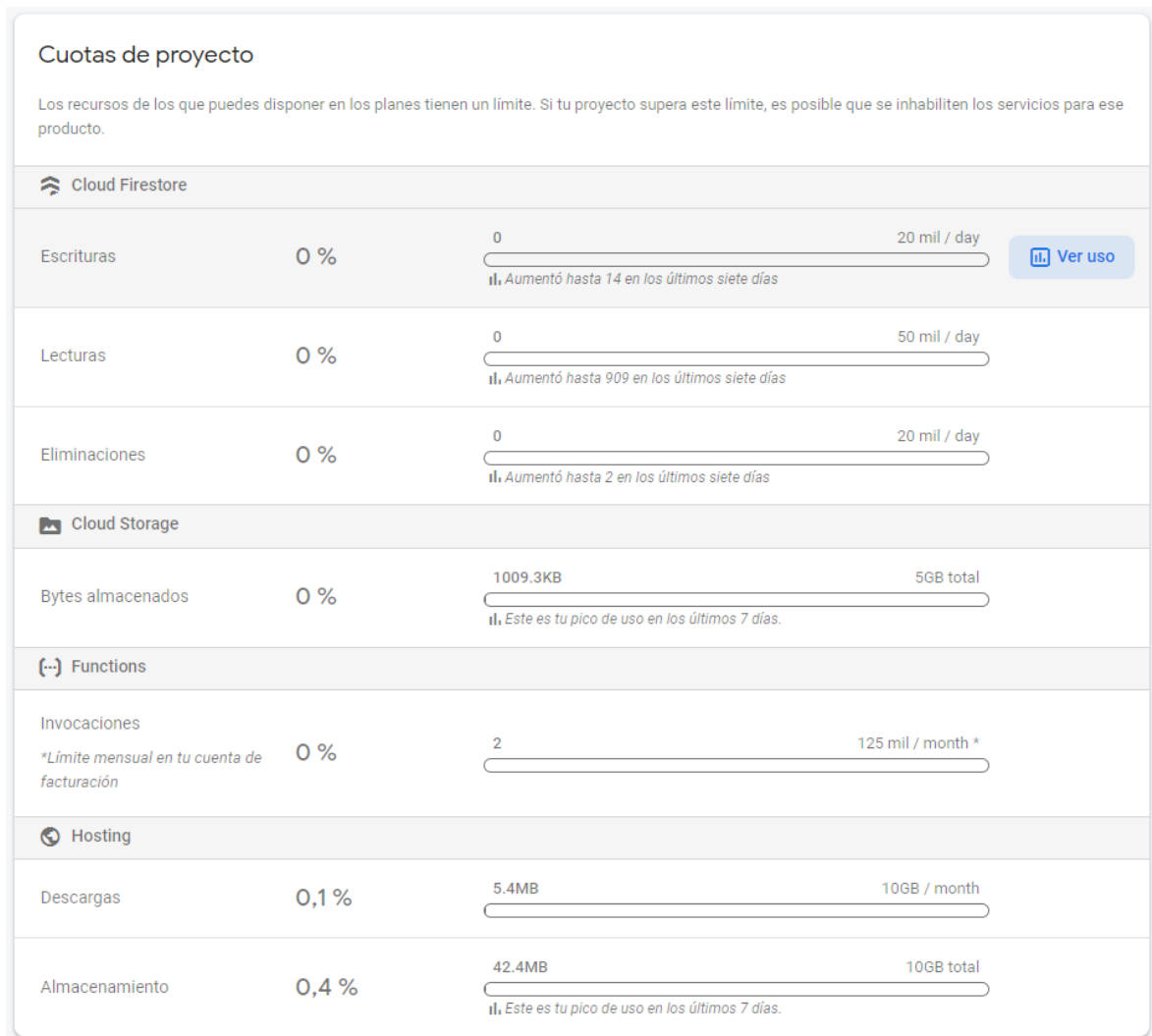


Ilustración 66: Desglose de las escrituras de Cloud Firestore

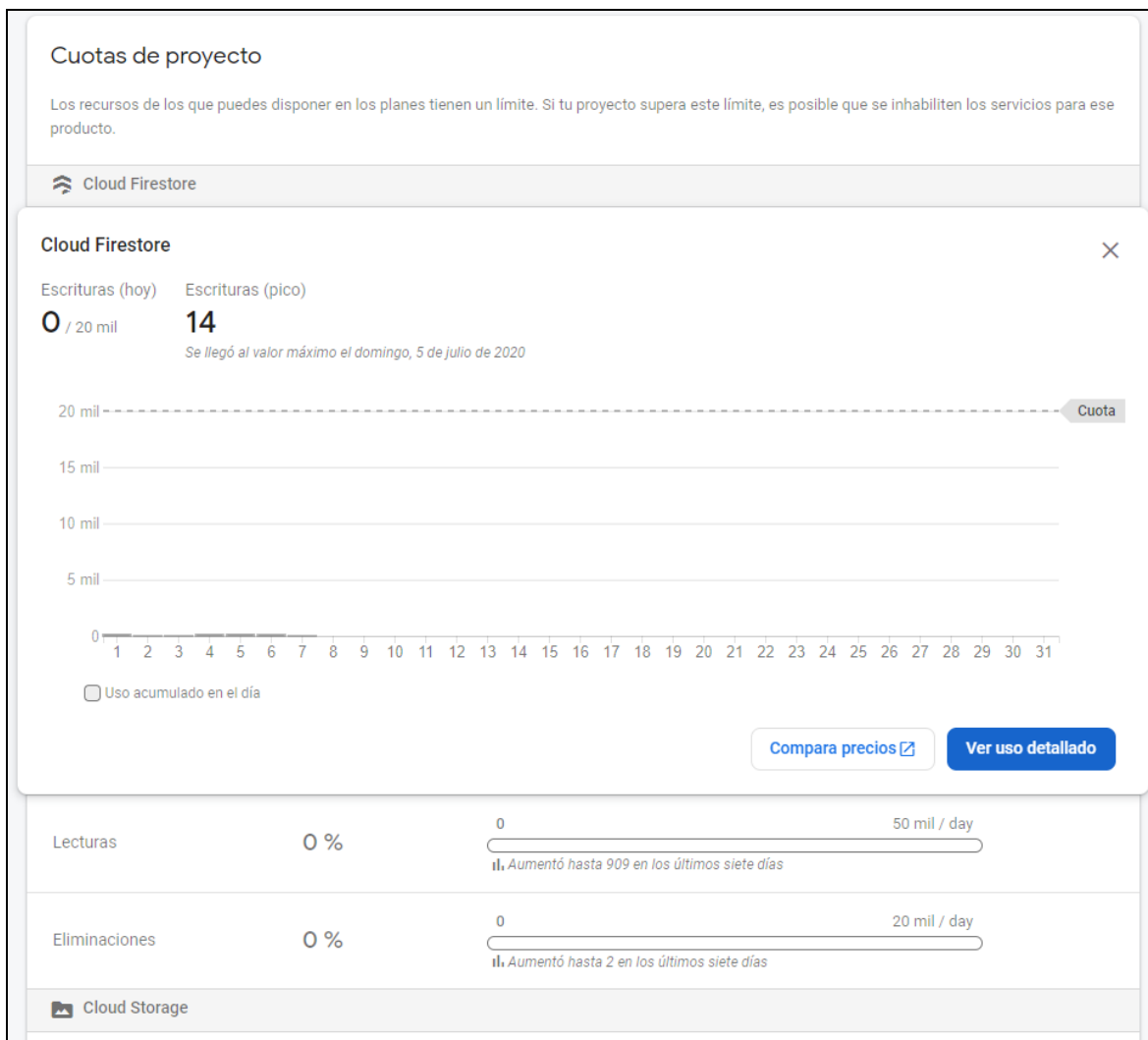


Ilustración 67: Uso detallado de escrituras en Cloud Functions (día actual)

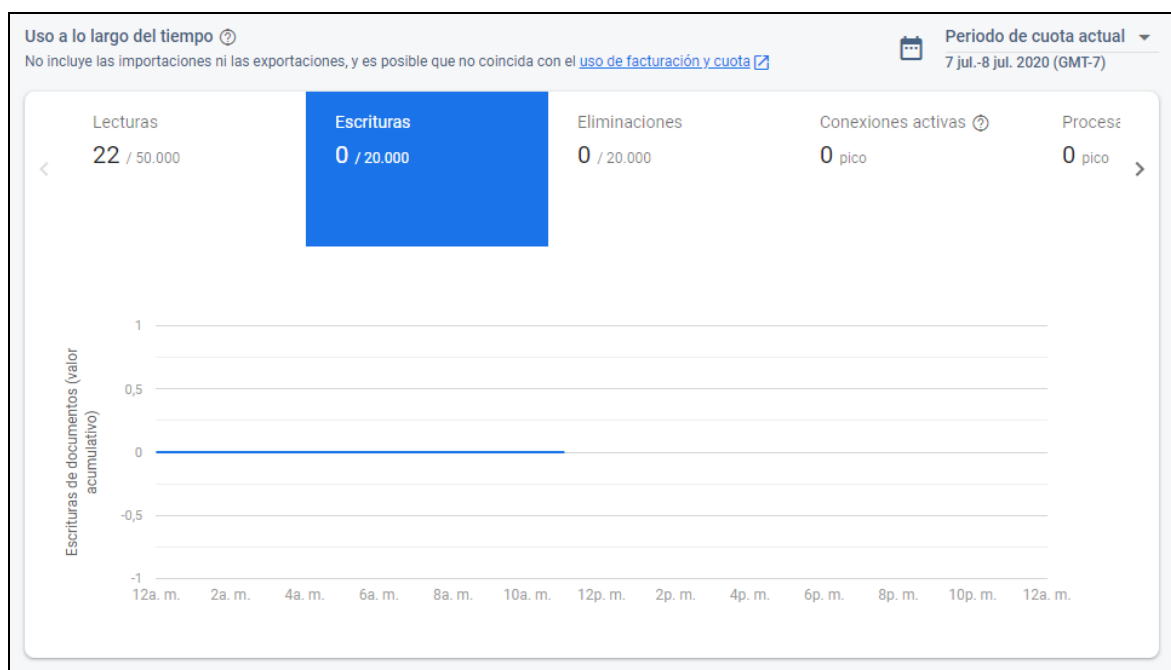


Ilustración 68: Uso detallado de escrituras en Cloud Functions (últimos 7 días)

